Technika mikroprocesorowa - *wykłady AGH*

Pieczołowicie spisał – Krzysztof Piecuch
Gorlice

19 października 2000 roku

Streszczenie

Dokument ten przedstawia wykłady z techniki mikroprocesorowej w roku akademickim 1995/1996

Spis treści

1 MIKROPROCESORY - wiadomości wstępne

Mikroprocesor możemy uważać za programowany automat sekwencyjny synchroniczny. Jego prawidłową pracę zabezpieczają sygnały zegarowe. Stosując mikroprocesor jako np. układ sterowania w przypadku zmiany algorytmu wystarcza tylko zmodyfikować jego program działania, natomiast jeżeli sterowanie rozwiążemy przy pomocy bramek przerzutników monostabilnych itp. niezbędna staje się przebudowa systemu tak więc, gdzie jest to tylko możliwe należy stosować rozwiązanie pierwsze. Mikroprocesor nie spełnia warunków tylko w przypadku, gdy konieczne jest równoczesne podjęcie wielu decyzji (program wykonuje się zawsze sekwencyjnie) oraz wymagana jest bardzo duża prędkość działania. Określa się częstotliwość górną zegarową (zawsze i dolną przeważnie), gdyż istnieją rozwiązania CMOS pozwalające na zatrzymanie sygnału zegarowego. Zależy to od typu użytych bramek logicznych w strukturze (funktory statyczne lub dynamiczne). Aby powstał system μP , oprócz samej jednostki centralnej należy dołączyć pamięć oraz układy we-wy. Pamięć dzieli się na pamięć programu i pamięć danych (ta pierwsza zawsze występuje). W pamięci programu znajduje się przepis działania μP - ma on postać zero-jedynkową. Zera i jedynki zorganizowane są w bajty. Od tego na ilu bitach naraz potrafi wykonywać operacje dany μP , mówimy o rozwiązaniach 4,8,16,24,32,64 bitowych.
Pod nazwą mikrokomputer (μC) rozumiemy cały system mikroprocesorowy. Scalenie takiego systemu do jednej struktury (CPU-32 bity Mb-pamięci) obecnie nie jest możliwe(rok 1996). Natomiast istnieje możliwość scalenia prostszych rozwiązań zawierających oprócz CPU, kilku kilobajtową pamięć programu, kilkusetbajtową pamięć danych oraz specjalizowane układy sprzętowe i linie we-wy.
CPU w μC jest zorientowany przeważnie dla potrzeb sterowania, a nie obliczeń numerycznych.
Tzw. układy sprzętowe to:

- liczniki, czasomierze
- różnego rodzaju interfejsy szeregowe, komunikacyjne, i2c
- przetworniki a/c MUX itp.

W momencie rozpoczęcia pracy przez μP zawartość pamięci programu musi odpowiadać programowi do wykonania (pamięć stała).
W przypadku μC możemy wykorzystywać wersję z:

1. wewnętrzną pamięcią EPROM lub PROM

2. wewnętrzną pamięcią ROM (programowaną na etapie masek u producenta)
3. pamięcią zewnętrzną (stałą)

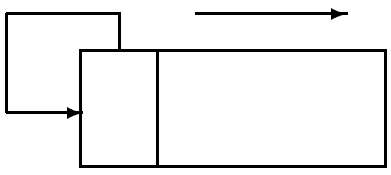
W przypadku rozwiązań prototypowych i małoseryjnych wykorzystujemy rozwiązania 1 i 2, natomiast można zamówić u producenta μC z własnym programem tylko wtedy, gdy będzie on całkowicie dopracowany oraz w przypadkach wieloseryjnych. μP dysponuje pewną ilością linii w ramach tzw. szyny adresowej. Ich zerojedynkowe kombinacje określają kolejne tzw. adresy. Zbiór wszystkich adresów tworzy tzw. przestrzeń adresową (2^n - gdzie n liczba bitów), tak więc dany μP może wyszczególnić (adresować) 2 tzw. słów. Przez słowo rozumiemy ilość bitów równocześnie wysyłaną bądź odbieraną przez dany μP . O rozmiarach słowa decyduje szerokość tzw. szyny danych.
Przestrzeń adresową podajemy zawsze w tzw. bajtach, tak więc dla szyny 16-bitowej "wypadnie" linia A0, a dla szyny 32-bitowej również A1.
Przestrzenie adresowe mogą być jednolite bądź rozdzielone. W przypadku przestrzeni jednolitej, wszystkie układy we-wy "podszywają się" pod pewne lokacje adresowe.μP każdą lokację w swej przestrzeni traktuje jako pamięć RAM. W przypadku przestrzeni rozdzielonych dla układów we-wy przewiduje się przestrzeń oddzielną, przeważnie zawsze mniejszą (max. 2) sterowaną przez oddzielne linie. Do kontaktu z tą przestrzenią są używane oddzielne rozkazy. W przypadku μC spotykamy również dwie podprzestrzenie pamięci danych zewnętrzną i wewnętrzną.
Przez cykl maszynowy rozumiemy jednokrotny kontakt μP z pamięcią bądź układem we-wy. Dany μP wykonuje swój program w cyklach maszynowych wtedy, gdy nie dysponuje tzw. kolejką rozkazów. Istnieje pełna korelacja pomiędzy cyklem maszynowym, a μP uprzednim działaniem μP . Jeśli istnieje tzw. kolejka możemy mówić tylko o pewnym cyklu magistrali. Pomiędzy cyklem magistrali, a działaniem μP przeważnie związek nie występuje. Do kolejki μP pobiera kolejne rozkazy do wykonania w czasie gdy magistrala i tak byłaby niewykorzystana.
Ponieważ w kolejno pobieranych instrukcjach mogą znaleźć się takie, które będą zaburzały sekwencyjność, więc w tym przypadku μP unieważnia kolejkę i rozpoczyna jej kompletowanie od nowa. Wprawdzie ilość cykli magistrali μP z kolejką jest zawsze większa od ilości cykli dla wersji równoważnej(bez kolejki) to jednak kolejka w istotny sposób przyspiesza działanie μP . Liczba cykli magistrali ulega natomiast wyraźnej redukcji jeżeli dany μP mimo mechanizmu kolejki dysponuje tzw. pamięcią podręczną (dla danych i programu - cache). Wykonując dany rozkaz bądź operację na danych tworzy się kopię używanych lokacji używanej przestrzeni adresowej wewnątrz

pamięci CACHE. Ponadto przed każdym cyklem magistrali μP sprawdza czy dana lokacja nie znajduje się w pamięci podręcznej, jeżeli tak to do cyklu magistrali nie dochodzi. Ponieważ duże odcinki programu zawierają tzw. pętle programowe, więc uzyskane tą drogą oszczędności czasowe są bardzo istotne (μP pobierze rozkazy tylko przy pierwszym nawrocie, a przy pozostałych będzie korzystał z pamięci podręcznej).

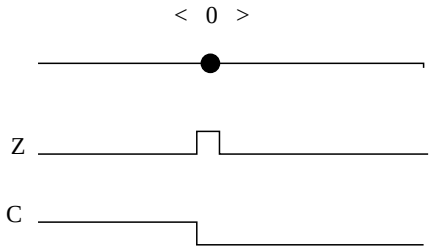
1.0.1 Wybrane zagadnienia zapisu binarnego

	+7	0	1	1	1
	6	0	1	1	0
	5	0	1	0	1
	4	0	1	0	0
	3	0	0	1	1
	2	0	0	1	0
+	1	0	0	0	1
-	0	0	0	0	0
	-1	1	1	1	1
	-2	1	1	1	0
	-3	1	1	0	1
	-4	1	1	0	0
	-5	1	0	1	1
	-6	1	0	1	0
	-7	1	0	0	1

W zapisie U2, aby zmienić znak liczby należy ją zanegować i dodać jedynkę. μP posiadają przeważnie oddzielne rozkazy do zmiany znaku, oraz rozkazy negacji. Rozkazy przesunięć logicznych i rotacji są oczywiste, wyjaśnienia wymagają jedynie instrukcje przesunięć arytmetycznych. **Przy przesunięciu w lewo na zwolniony najmłodszy bit wsuwa się zawsze zero, natomiast przy przesunięciu arytmetycznym w prawo następuje powielenie bitu znaku, rys 1**
Powyższe zależności obowiązują dla kodu U2. W wyniku operacji arytme-



Rysunek 1: Powielenie bitu znaku



tycznych i logicznych μP ustawia odpowiedni stan bitów warunkowych:

Z - zerowość

C - przeniesienie (carry) (CY)

V - przekroczenie zakresu (znaku)

P - parzystość

AC - przeniesienie pomocnicze

Bit Z jest ustawiany wtedy, gdy w wyniku operacji otrzymaliśmy zero (np. $-5-(-5)$**).**

7	0	1	1	1	
3	1	1	1	1	$\rightarrow$ CY=0
10	1	0	1	0	

7	0	1	1	1	
10	1	0	1	0	$\rightarrow$ CY=1
17	0	0	0	1	

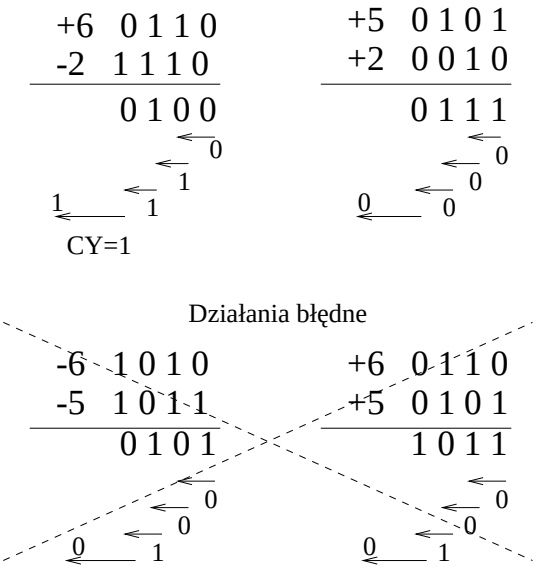
7	0	1	1	1	
-3	0	0	1	1	
4	0	1	0	0	

7	0	1	1	1	
-10	1	0	1	0	$\rightarrow$ pożyczka
10	0	1	0	1	

1.0.2 Relace pomiędzy liczbami bez znaku

$$A?B$$

$<$	C
$\geqslant$	$\overline{C}$
$=$	Z
$\leqslant$	$C \vee Z$
$>$	$\overline{C \vee Z}$



1.0.3 Relace pomiędzy liczbami ze znakiem

$$V = C_n * \overline{C_{n-1}} \vee \overline{C_n} * C_{n-1} = C_n \bigoplus C_{n-1}$$

V - pokazuje błędny wynik

	$<$			
	$\|A\| + \|B\| > K$ $A < 0$ $B > 0$	$\|A\| + \|B\| \leqslant K$ $A < 0$ $B > 0$	$A < 0$ $B < 0$	$A > 0$ $B > 0$
Z	0	0	0	0
N	0	1	1	1
C	1	1	0	0
V	1	0	0	0

Tabela 1: Bity warunkowe dla liczb ujemnych

	$=$		
	$A = B > 0$	$A = B = 0$	$A = B < 0$
Z	1	1	1
N	0	0	0
C	1	0	1
V	0	0	0

Tabela 2: Bity warunkowe dla liczb równych

	$>$			
	$A>0$ $B>0$	$A<0$ $B<0$	$\|A\|+\|B\| \leqslant K$ $A>0$ $B<0$	$\|A\|+\|B\|>K$ $A>0$ $B<0$
Z	0	0	0	0
N	0	0	0	1
C	1	1	0	0
V	0	0	0	1

Tabela 3: Bity warunkowe dla liczb dodatnich

$=$	Z
$\neq$	$\overline{\text{Z}}$
$<$	$\text{N} \bigoplus V$
$\geqslant$	$\text{N} \bigoplus V$
$\leqslant$	$\text{N} \bigoplus V \vee Z$
$>$	$\text{N} \bigoplus V \vee Z$

1.0.4 Zapis w kodzie BCD

BIN	HEX	DEC
01011101	5D	93

58		01011000
35	+	00110101
		10001101
		00000110 $+6 \leftarrow$ rozkaz korekcji dziesiętnej
		10010011

	1011	1101
+		0110
	0110	
	0010	0011
		$\leftarrow CY = 1$
	0110	
	1000	0011

Przy operacjach na liczbach dziesiętnych kodowanych binarnie otrzymujemy nie zawsze prawidłowe wyniki. Otrzymany wynik należy *"potraktować"* rozkazem korekcji dziesiętnej. Polega on na dodawaniu na odpowiednich pozycjach liczby 6 w przypadku liczby większej od 9 i w przypadku wystąpienia przeniesienia połówkowego (z poprzedniej pozycji). Niektóre μP nie posiadają rozkazu **DAA**, lecz mają możliwość przestawienia swojego arytmometru na pracę dziesiętną np: 6502.

	83		83	
−	58	dopełnienie do 100 →	42	
	25		25	(CY=1)

2 MIKROPROCESORY 8-bitowe

Będą rozpatrywane na przykładzie układów **6800, 6502, 8080, Z80**.[1] Typowa architektura μP zawiera następujące podzespoły:

1. licznik programu PC
2. jednostkę arytmetyczno-logiczną ALU
3. rejestr bitów statusowych (flagowych), których stan zmienia się wskutek działania ALU
4. akumulator (akumulatory)
5. wskaźnik stosu SP
6. rejestry ogólnego przeznaczenia
7. pomocnicze rejestry adresowe (np. indeksowe)
8. układy sterująco-kontrolne
9. układy magistral wewnętrznych

Po włączeniu napięcia zasilania do μP , należy podać sygnał RESET, który ustawia początkowy stan μP .Rozkaz RESET inicjuje również właściwy dla danego μP adres początkowy. Istnieją dwa sposoby wyznaczenia adresu pierwszego bajtu programu:

- zawartość PC wskazuje bezpośrednio na pierwszy bajt (kod operacyjny) programu
- adres początkowy wskazuje odpowiednie lokacje (dwie kolejne) w obszarze pamięciowym pod którymi jest zapisany adres pierwszej instrukcji.

Pierwszą metodę spotyka się u INTELA i pochodnych, drugą w MOTOROLI i pochodnych. μP łączy się z układami pamięciowymi oraz we/wy za pomocą tzw. *magistrali* tzn. zbioru odpowiedzi zdefiniowanych linii.
Wśród nich wyróżniamy tzw. *szynę adresową*, *danych*, oraz *szyny sygnałów sterujących*. Szyna adresowa jest jednokierunkowa trójstanowa, a szyna danych dwukierunkowa - trójstanowa. Cykl instrukcji jest to wykonanie przez μP kolejnego rozkazu. Na cykl instrukcji składa się jeden lub więcej cykli maszynowych. Cykl rozkazu rozpoczyna się zawsze od pobrania kodu operacyjnego kolejnego rozkazu do wykonania. Większość μP dysponuje możliwością tzw. zawieszenia. W tym stanie μP zatrzymuje swoje

[1] J.Kaczmarczyk *μP Z80*,
J.Kaczmarczyk H.Kruszyński *Mikroprocesor 6502 i jego rodzina*,
K.Fedyna M.Mizeracki *Układy mikroprocesorowe Z80*

działanie, wprowadza swoje linie w stan nieaktywny bądź wysokiej impedancji, a o fakcie tym informuje na zewnątrz odpowiednią linią sygnałową. W czasie zawieszenia μP inny układ lub inny μP może uzyskać dostęp do pamięci systemu.
Ilość cykli maszynowych z których składa się program jest zawsze równa lub większa od ilości bajtów programu. Każdy cykl maszyn zaczyna się wystawieniem adresu, adres też jako ostatni zostanie wycofany. Następnie w przypadku transmisji na zewnątrz, aktywuje szynę danych, a ich ważność potwierdza strobem zapisu. W przypadku transmisji do μP aktywuje strob odczytu i ustawia do wewnątrz szynę danych. Jeśli właściwy dla μP czas cyklu maszynowego jest zbyt krótki, dane urządzenie bądź pamięć może zażądać jego wydłużenia poprzez wstawienie taktów oczekiwania (μP jest w stanie aktywnym). Ilość wstawianych taktów oczekiwania nie jest niczym limitowana. Metodę tę można wykorzystać do sprzętowej pracy krokowej.
W przypadku konieczności reakcji μP na niesynchroniczne zdarzenia zewnętrzne, używa się tzw. techniki przerwań. Sygnał przerwania powoduje, że μP zatrzymuje wykonywanie aktualnego programu kończąc bieżący rozkaz, zapamiętuje adres następnego rozkazu, tego który byłby wykonany gdyby nie przerwanie i wykonuje obsługę przerwania. Obsługa ta polega na realizacji pewnego programu, po jego zakończeniu μP powraca do tej instrukcji, której adres zapamiętał w momencie akceptacji przerwania.
Jeżeli np. w momencie rozpoczynania CM (cyklu maszynowego) przez μP pojawią się trzy sygnały:

- niegotowości
- zawieszenia
- przerwania

to reakcja μP będzie następująca:

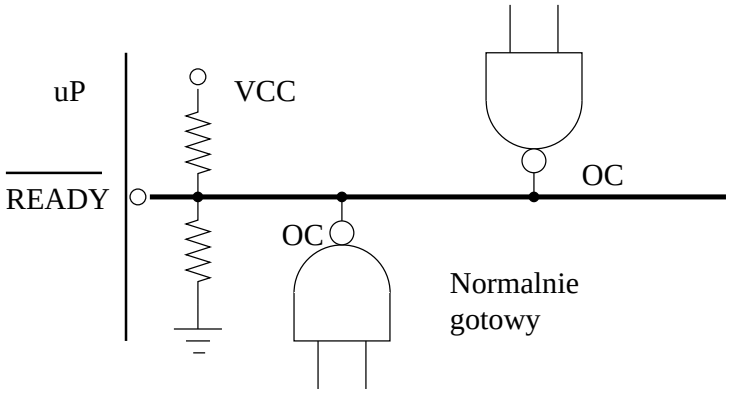
1. aż, do wycofania linii niegotowości μP będzie wstawiał takty oczekiwania
2. zakończywszy CM zaakceptuje zawieszenie i odda magistralę
3. po wycofaniu żądania zawieszenia zaakceptuje przerwanie wtedy, gdyby cykl ten był ostatnim cyklem rozkazu, czyli jeżeli nie to najpierw dokończy brakujące cykle.

Każdy układ bądź blok pamięci w systemie posiada swój adres bądź blok kolejnych adresów. W momencie pojawienia się wyróżnionego adresu bądź układów we/wy stwierdzają, że aktualny CM ich dotyczy. Należy zaznaczyć, że przestrzeń adresowa nie musi być całkowicie obsadzona (na pewne adresy nikt nie odpowiada). Należy pamiętać także, aby pod tym samym

adresem nie ulokować dwóch lub więcej układów, gdyż uniemożliwi to poprawną pracę systemu.
Systemy μP można podzielić na systemy normalnie:

- gotowe (linia niegotowości, rys. 2)
- niegotowe (linia potwierdzenia, rys. 3)

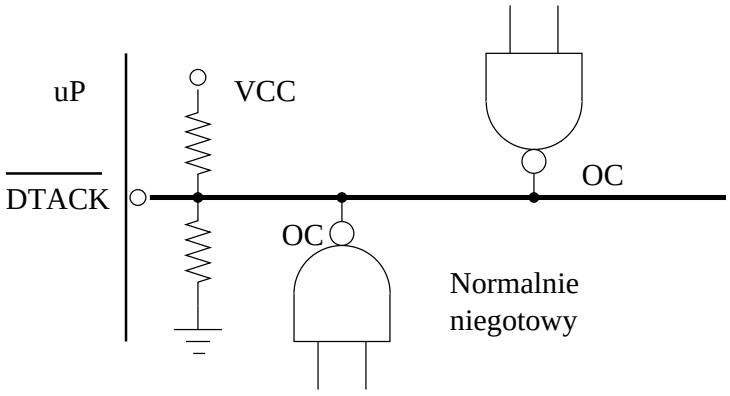
W systemie normalnie gotowym podzespoły, które nie wymagają wydłużenia CM nie ingerują w linie gotowości, która pozostaje w swoim stanie aktywnym. W wypadku kontaktu z nieobsadzoną lokacją μP nie jest w stanie tego faktu stwierdzić (zapisane dane ulegają straceniu, w przypadku odczytu pojawi się bajt FF potraktowany jako dane bądź kod operacyjny rozkazu).
W systemie normalnie niegotowym linia potwierdzenia transmisji znajduje



Rysunek 2: System normalnie gotowy

się w normalnie wysokim nieaktywnym stanie.
Każdy podzespół po zaadresowaniu musi potwierdzić transmisję (układy szybkie natychmiast, wolniejsze odpowiedzą później). Przy zaadresowaniu lokacji nieobsadzonej nie będzie miał kto potwierdzić i μP będzie wstawiał takty oczekiwania nie kończąc cyklu. Fakt ten może być wykryty przez zewnętrzne układy tzw. przeterminowania, które zgłoszą do μP fakt zaistnienia tzw. błędu magistrali (wymuszone zakończenie cyklu i odpowiednia obsługa).

Tak więc system normalnie niegotowy jest w stanie wykryć kontakt z nieobsadzonym adresem. Aby to było możliwe, każdy moduł systemu normalnie niegotowego musi generować sygnał potwierdzenia transmisji, jeśli stwierdzi na szynie adresowej swój ważny adres.



Rysunek 3: System normalnie niegotowy

Porównując cztery typy 8- bitowych μP stwierdzamy, że każdy z nich adresuje 2^{16} bajtów z tym, że 8080 i Z80 dodatkowo może adresować oddzielną przestrzeń układów we/wy. Jest to 256 lokacji dla 8080 i $2^8(2^{16})$ lokacji dla Z80. Przestrzeń we/wy nie występuje dla μP 6800 i 6502. Układy we/wy muszą być w tych systemach ulokowane w pamięci, tzn. "podszywać się" pod odpowiednie w niej lokacje. Mówimy, że μP posiada rozdzieloną bądź jednolitą przestrzeń adresową. Każdy μP posiada jedną bądź dwie fazy zewnętrznego sygnału zegarowego (określona górna i dolna jej wartość), za wyjątkiem 6800 pozostałe μP mają możliwość wydłużenia CM (przystosowane w systemach normalnie gotowych). μP 6502 nie posiada możliwości zawieszenia i oddania magistrali, w pozostałych układach można to przeprowadzić.
Rozwiązanie Z80 i 8080 (linia żądania i potwierdzenia) przyjęło się , natomiast rozwiązanie 6800 (czas dostępu limitowany po każdym cyklu od linii TSC lub czas nielimitowany po każdym rozkazie od linii HALT) nie znalazł kontynuacji. Za wyjątkiem 8080 pozostałe μP oprócz normalnego przerwania (można je blokować programowo) posiadają też wejście przerwania niemaskowalnego NMI, które nie może być zablokowane programem.
Także za wyjątkiem 8080 wszystkie μP dysponują liniami sterującymi w sposób jawny (8080 zatrzaskuje te sygnał na początku cyklu,kiedy pojawiają się na niewykorzystywanej jeszcze szynie danych). μP posiadające rozdzieloną przestrzeń dysponują oczywiście liniami, które pozwolą na rozróżnienie kontaktów z liniami we/wy.
W zależności od przeznaczenia cykle możemy podzielić na następujące rodzaje:

- pobranie (kodu operacji , rozkazu)

- odczyt pamięci
- zapis pamięci
- odczyt wejścia
- zapis wyjścia
- akceptacja przerwania, rys. 4

Po zaakceptowaniu przerwania μP wykonuje jego cykl akceptacji. W cyklu tym oczekuje od urządzenia przerywającego, aby poprzez szynę danych dostarczyło albo kodu operacji - pierwszego rozkazu obsługi przerwania - albo numeru wektora. Z numerem wektora w tablicy wektorów jest związany odpowiedni adres od którego zaczyna się procedura obsługi przerwania. W przypadku, gdy μP otrzymuje KO (kod operacji) musi być to kod instrukcji wywołania.

		8080, 8086	Z80	6800, 6502	
Pobranie					
Pamięć	odczyt				
	zapis				
we/wy	odczyt				
	zapis				
akceptacja przerwania					

Tabela 4: Wykaz sygnałów biorących udział w operacjach na pamięci, portach we/wy, przerwaniach w zależności od typu procesora

---

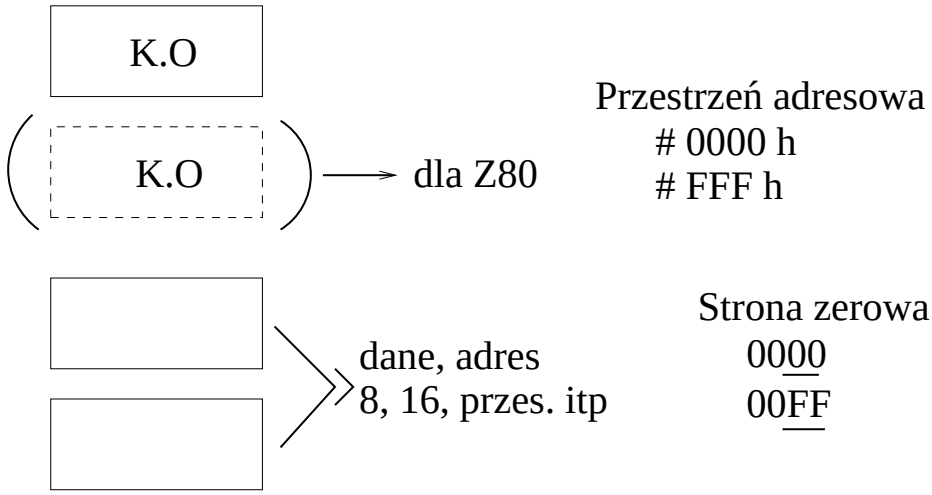
μP 6800 i 6502 nie potrzebują także, cykli akceptacji przerwania, gdyż w odpowiedzi na nie sięgają zawsze pod stałe lokacje przestrzeni, gdzie znajdują adresy początków obsługi przerwania zwykłego IRQ i niemaskowalnego NMI. μP 6800 oprócz tego, że pracuje w sposób synchroniczny (brak wstawianych taktów oczekiwania) wykonuje także tzw. puste cykle. W cyklach tych nie jest aktywna VMA. Cykle te są potrzebne na dokończenie wewnętrznych operacji (arytmetyczne). Po wykonaniu każdego rozkazu licznik PC jest zwiększany o wartość od 1 do 4 tzn. o taką na ilu bajtach zapisany był poprzednio wykonywany rozkaz. Sekwencyjność ta jest zaburzana w przypadku wykonywania rozkazu skoków, wywołań i powrotów. Wykonując skok μP wpisuje do PC inną wartość zgodnie z wskazanym trybem adresacji. Po przejściu do nowego adresu tracony jest wszelki ślad z poprzednim fragmentem programu. W przypadku rozkazu wywołania μP pamięta na tzw. stosie tą wartość PC spod której pobrał by kolejny rozkaz,

gdyby nie istniała instrukcja wywołania. Fragment programu (podprogram) do którego można wielokrotnie z różnych miejsc odwoływać się instrukcją wywołania jest zakończony rozkazem powrotu. Rozkaz ten odtwarza ze stosu zapamiętaną wartość PC i μP powraca do miejsca skąd nastąpiło wywołanie. Technikę tę stosujemy w przypadku konieczności wykonywania w różnych miejscach programu głównego pewnego stałego zbioru operacji (podprogram).
W przypadku skoku mówimy o skokach bezwarunkowych i warunkowych. Skok bezwarunkowy wykonywany jest zawsze, ewentualne wykonanie skoku warunkowego zależy od zadeklarowanego w kodzie operacyjnym stanu bitów warunkowych. Jeżeli warunek nie jest spełniony program wykonuje się dalej. Rozkazy warunkowe wywołań i powrotów występowały w 8080 i Z80, w następnych konstrukcjach nie znalazły zastosowania. Porównując architektury omawianych μP zauważamy, że każdy z nich dysponuje 16-bitowym rejestrem PC oraz SP. Rejestry te mają taką samą szerokość jak szyny adresowe i służą do adresowania pamięci przy pobieraniu instrukcji i do adresowania wyróżnionego w niej obszaru, gdzie znajduje się tzw. stos. Wyjątkiem jest 6502, którego 8 starszych bitów rejestru SP są określone na stałe (stos może zawierać tylko 256 lokacji). Ponadto każdy układ posiada wewnętrzny blok ALU i związane z nim bity warunkowe zgrupowane w tzw. rejestrze bitów warunkowych (statusowych, flagowych).
Z układem ALU jest także związany wyróżniony rejestr tzw. akumulator, który jest argumentem większości operacji arytmetycznych i logicznych. Ponadto architektura wewnętrzna może zawierać pewną ilość rejestrów (występujących oddzielnie jako 8-bitowe, bądź zespołowo jako rejestry 16-bitowe). Rejestry te są wykorzystywane np. w trybie adresacji indeksowej (X, Y, IX, IY), bądź mają własności uniwersalne (źródło adresu, względnie dane). Rejestry uniwersalne występują w 8080, a Z80 posiada dwa alternatywne ich zestawy. Analizując zawartość bitów warunkowych omawianych μP zauważamy, że wszędzie występują bity znaku, przeniesieniai zerowości oraz przekroczenia zakresu za wyjątkiem 8080. Jeśli arytmometr danego μP (6502) możemy przełączyć na pracę dziesiętną (bit D) to zbędnym staje się bit warunkowy przeniesienia pomocniczego. Bit parzystości występuje tylko w 8080 i Z80, w którym ma podwójne znaczenie parzystość po operacjach logicznych, przekroczenie zakresu w przypadku operacji arytmetycznych). Począwszy od 6800 (też 6502) wprowadzono do rejestru bitów warunkowych bit zezwalający bądź maskujący przerwanie. Bit ten jest konieczny μP jeśli muszą ze sobą współistnieć przerwania maskowalne i niemaskowalne. μP Z80 w tym celu zastosował przerzutniki IFF1 oraz IFF2. Możemy zauważyć, że architektura Z80 jest identyczna z 8080. Zostały tylko wprowadzone rejestry alternatywne i indeksowe, a także rozbudowano system przerwań

(rejestr I). μP Z80 wykonuje z drobnymi wyjątkami postać binarną programu generowaną dla 8080. Zgodność na poziomie kodu. Jeden bajt pozwala na rozróżnienie przez dekoder instrukcji μP 256 kodów operacyjnych (rozkazów), zdarza się, że pewna część tych kodów nie jest wykorzystywana (μP z otwartą listą rozkazów). Podanie takiej kombinacji jako KO rozkazu może spowodować:

- przypadkowe działanie
- wykonywanie instrukcji pustej
- zgłoszenie przerwania wewnętrznego (wyjątki)

W omawianych μP za wyjątkiem wersji 6502, 65C02 (instr. puste) spotyka się pierwszy wariant. Jeśli okaże się, że liczba rozkazów przewyższa szerokość słowa danych (tutaj bajt) to pewne kombinacje kodowe dają początek nowym 256- liczbowym rodzinom rozkazów. W tym przypadku na liście rozkazów μP koegzystują ze sobą rozkazy 1 i 2 - bajtowe. Metodę tę przyjęto dla Z80, który wykonuje kod z 8080, ponieważ liczba wolnych kombinacji kodowych dla 8080 wynosiła tylko 12 więc dla Z80 musiano też wprowadzić rozkazy kodowane na 2-bajtach.
W pewnych rozwiązaniach w przypadku μC dokonuje się optymalizacji rozkazu. Polega ona na tym, że często występujące instrukcje mają krótki, a rzadko występujące dłuższy kod operacyjny.

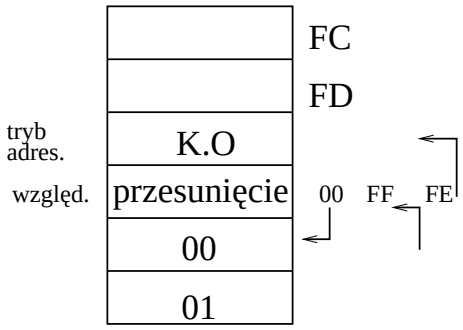


Nazwa		**8080**	**Z80**	**6800**	**6502**	
Implikacyjne, *implied*		+	+	+	+	1
Natychmiastowe, *immediate*		+	+	+	+	2
Rejestrowe, *register*		+	+			3
Rejestrowe pośrednie,*register indirect*		+	+			4
Bezpośrednie	krótkie			+	+	5
Direct	pełne		+	+	+	
Względne	krótkie		+	+	+	6
Relative	pełne					
Indeksowe	krótkie				+	7
Indexed	pełne		+	+	+	
Pośrednie	krótkie				+	8
Indirect	pełne				+	
Indeksowe pośrednie	krótkie				+	9
Indexed indirect	pełne				+	
Pośrednie indeksowe	krótkie				+	10
Indirect indexed	pełne					
Bitowe, *bits*			+			11

Tabela 5: Tryby adresacji μP

Im większą ilością trybów (rys. 5 adresowania dysponuje μP tym łatwiej i elastyczniej można stworzyć zadane oprogramowanie. Tryb adresacji podaje sposób wyznaczenia argumentu. Dla trybu od 4 - 11 za wyjątkiem 6, dotyczy to przestrzeni adresowej pomiędzy lokacjami 0000 a FFFF (2^{16} bajtów). Dla μP 6800 i 6502, które nie posiadają wewnętrznych uniwersalnych rejestrów wyznaczono tzw. stronę zerową (pierwszych 256 lokacji). Rozkazy dotyczące strony zerowej stosują skrócony 1-bajtowy adres. Adresacja implikacyjna dotyczy przypadku, kiedy argument wynika wprost z rozkazu np. zaneguj akumulator. Przy adresacji natychmiastowej argument wpisuje się wprost za kodem operacyjnym (jeden bądź dwa bajty). Jeśli argument znajduje się w rejestrze lub parze rejestrów mamy doczynienia z adresacją rejestrową. W przypadku kiedy rejestr (para rejestrów) nie zawiera argumentu lecz tylko jego adres mamy doczynienia z adresacją rejestrową pośrednią. Jeśli jakieś źródło adresu nie korzysta z reje-

stru lecz adres wpisuje bezpośrednio w programie za kodem operacyjnym to mówimy o adresacji bezpośredniej. Adresacja względna (rys. 4 podaje natomiast adres w odniesieniu do aktualnej zawartości licznika programu PC. Ten tryb adresacji wykorzystują instrukcje skoków i wywołań. *Jeśli w programie przy skokach i wywołaniach będziemy stosowali powyższy tryb uzyskamy własność relokowalności kodu tzn. program będzie mógł być umieszczony w dowolnym miejscu pamięci.*



Rysunek 4: Tryb adresacji względnej

Omawiane μP dysponują jedynie jego krótką wersją, która umożliwia skoki wstecz o 128 bajtów i w przód o 127 bajtów. W przypadku adresacji indeksowej do adresu w rozkazie dodawana jest w kodzie U2 zawartość rejestru indeksowego tworząc dopiero adres argumentu. O ile przy adresacji bezpośredniej adres argumentu był wprost za KO o tyle w adresacji pośredniej adres w rozkazie wskazuje na dwie kolejne lokacje w przestrzeni adresowej, pod którymi znajduje się dopiero właściwy adres argumentu. Wersja indeksowa adresacji pośredniej dolicza do adresu za kodem operacyjnym zawartość rejestru indeksowego i dopiero ten adres wskazuje na dwie kolejne lokacje z adresem docelowym. Trybu tego nie należy mylić z adresacją *pośrednią indeksową*, w której zwartość rejestru indeksowego jest dodawana na drugim etapie wyznaczania adresu. O ile poprzedni tryb jako argumentu używał całego bajtu znajdującego się we wnętrzu μP bądź zewnętrznej przestrzeni adresowej, o tyle adresacja bitowa pozwala na zlokalizowanie pojedynczego bitu w rejestrze bądź pamięci zewnętrznej.

2.0.5 Wersja angielska nazewnictwa

1. implied
2. immediate
3. register
4. register indirect
5. direct
6. relative
7. indexed

8. indirect
9. indexed indirect
10. indirect indexed
11. bits

Linie przerwań pozwalają na asynchroniczną reakcję μP na zdarzenia zewnętrzne. Rozróżniamy przerwania maskowalne i niemaskowalne. Przerwania NMI są akceptowane zawsze. Przerwanie maskowalne μP dopuszcza pod warunkiem, że bit maski bądź zezwalający ma odpowiedni stan (odpowiednio bity I dla 6800 i 6502 oraz bit IFF1 dla Z80) ponadto dla obu przerwań warunkiem akceptacji jest zakończenie instrukcji. **Ponieważ NMI jest przyjmowane zawsze, dlatego każdy μP akceptuje je w momencie podania wyróżnionego zbocza**. Zapobiega to przerwaniu NMI przez to samo NMI, gdybyśmy przyjęli akceptację wyróżnionym poziomem sygnału. **Przerwanie INT (SRQ) są akceptowane wyróżnionym poziomem**, aby przerwanie nie przerwało samego siebie μP po jego zaakceptowaniu z "urzędu" ustawia odpowiednią wartość bitu **I** (IFF1) zabraniając sobie dalszych akceptacji tego przerwania. Ponieważ każde żądanie wycofuje się po obsłużeniu, więc po powrocie do programu głównego możemy z powrotem odblokować przerwanie maskowalne. Przerwania te mogą być odblokowane wyłącznie programowo (odpowiednim rozkazem). Ich zablokowanie dokonywane jest oprócz powyższego przypadku także programowo oraz zawsze po wyzerowaniu μP . W przypadku jeśli spodziewamy się przerwania NMI pierwszym rozkazem μP będzie instrukcja ustawiająca SP. Dla omawianych μP jeśli podczas obsługi NMI pojawi się następne aktywne zbocze to NMI przerwie samo siebie. Począwszy od μP 16 - bitowych wprowadzono odpowiednią blokadę, aktywne z bocze jest pamiętane i czeka na zakończenie poprzedniej obsługi przerwania NMI. Przy równoczesnym stwierdzeniu obu przerwań μP zaakceptuje oczywiście NMI (można oczywiście odblokować je programowo, lecz mija się to z celem). Na czas obsługi przerwania NMI należy zapamiętać czy przed jego akceptacją mogły być dopuszczone zwykłe przerwania. Dla 6800 i 6502 sprawa wygląda prosto.
Po wejściu w każde przerwanie na stosie zostaje położony oprócz PC także stan rejestrów warunkowych. W związku z tym rozkaz powrotu z przerwania (identyczny dla NMI) i maskowalnego przywróci poprzedni stan przyzwolenia na przerwanie maskowalne. Dla Z80, gdzie o przerwaniu decyduje stan IFF1 wprowadzono drugi przerzutnik IFF2, który przechowuje stan zezwolenia podczas obsługi NMI. Taki system niesie ze sobą następujące konsekwencje:

1. μP posiada dwa rozkazy powrotu, z przerwania (z podprogr.) i powrotu z obsługi NMI, tylko drugi kopiuje IFF2 do IFF1
2. Ponieważ po wejściu w zwykłe przerwanie zerowany jest IFF1, więc aby przywrócić akceptację tych przerwań należy w procedurze przerwania przed rozkazem powrotu umieścić instrukcję odblokowującą przerwanie maskowalne. Mimo, iż następne zgłoszenie może już czekać μP przed jego akceptacją wykonuje po EI jeszcze następny rozkaz i dopiero akceptuje przerwanie

μP po zatwierdzeniu przerwania wykonuje tzw. cykl akceptacji przerwania. W cyklu tym na szynie danych oczekuje pewnych informacji od urządzenia, które przerwało mu pracę. Informacją tą może być:

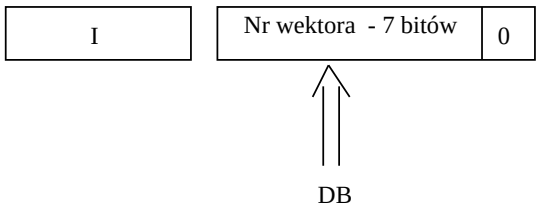
1. kod operacji pierwszego rozkazu obsługi przerwania
2. nr wektora (adresu początku obsługi) z odpowiedniej tablicy

Wersja 1 występuje dla 8080 i zerowego trybu przerwań Z80. W dosyłanych cyklach akceptacji 1 rozkaz podprogramu obsługi musi spełniać następujące wymogi:

- mieści się w jednym bajcie (cykl akceptacji jest tylko jeden)
- zachowuje ślad powrotu na stosie
- skierowuje μP w inne miejsce

Warunki te spełniają wyłącznie rozkazy restartu RST (n: 0 - 7)
Programy obsługi zaczynają się odpowiednio od lokacji 0,8,10...38H.
Jeśli dany μP dysponuje tylko jednoelementową tablicą wektorów (adresów początkowych) to cykl akceptacji staje się zbędny. Sytuacja taka ma miejsce dla 6800 i 6502 (lokacje na końcu przestrzeni adresowej zawierają adresy początkowe obsługi przerwania NMI i maskowalnego). Dla μP Z80 pracującym w trybie drugim obsługi przerwania tablica wektorów zawiera 128 elementów (zajmuje 256 bajtów). Jej ulokowanie w przestrzeni adresowej wskazuje rejestr I. W cyklu akceptacji dosyłane od urządzenia przerywającego poprzez szynę danych numer wektora tworzy wraz z zawartością rejestru I 16-bitowy adres, rys. 5. Adres ten definiuje w tablicy wektorów dwa sąsiednie bajty na których zapisany jest adres początku obsługi przerwania. Tak więc akceptacja w trybie drugim składać się będzie z 5- cykli maszynowych:

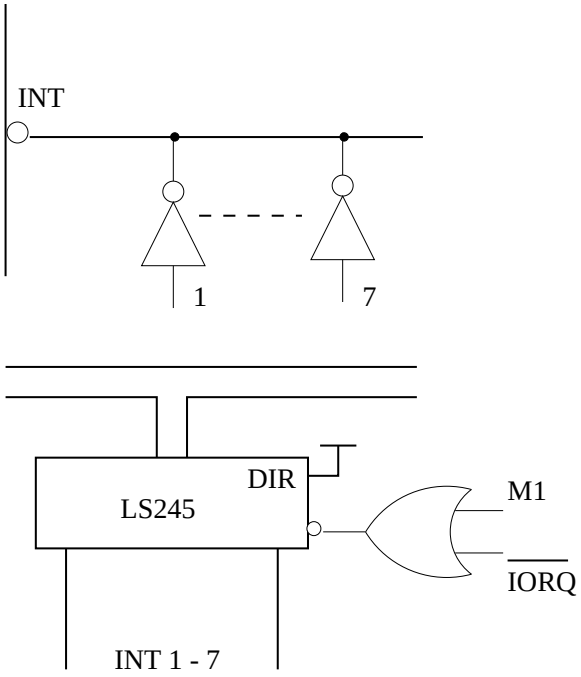
1. odczyt nr wektora
2. zapis stosu
3. zapis stosu
4. odczyt adresu początku obsługi przerwania



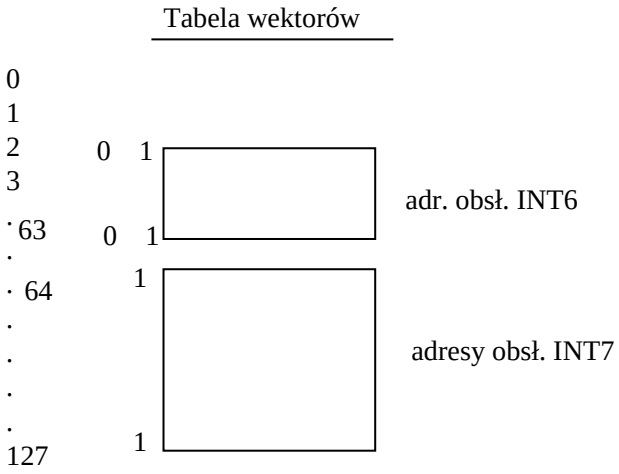
Rysunek 5: Tworzenie adresu w trybie drugim μP Z80

5. odczyt adresu początku obsługi przerwania

Występujący jeszcze w Z80 tryb pierwszy jest przypadkiem trybu zerowego ograniczonego do RST 7. Rozkaz ten jest pobierany wewnętrznie przez μP w odpowiedzi na jakiekolwiek przerwanie maskowalne (układy przerywające są zwolnione od podawania w cyklu akceptacji do μP kodu restartu). W 6502 i 6800 ustalenie początku obsługi NMI jest dokonywane konsekwentnie metodą wg. punktu 2. W Z80 przyjął się mało elastyczny sposób rozpoczynania przerwania NMI od stałego adresu 0066H.

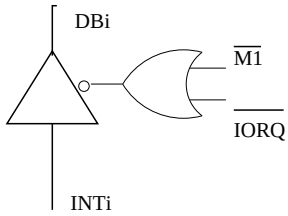


Rysunek 6: System obsługi priorytetowej przerwań procesora Z80



Rysunek 7: Tablica wektorów

Powyższy przykład ilustruje jak wykorzystując tryb 2 przerwań Z80 można stworzyć dla 7 źródeł przerwań automatyczny system obsługi priorytetowej, rys. 6, 7. Pojedynczy układ bufora trójstanowego może też być w zależności od sytuacji dokomponowany na poszczególnych kartach systemu do pojedynczej bramki trójstanowej.



Rysunek 8:

W miejscu, gdzie jest ulokowana tablica wektorów znajduje się zwykle pamięć typu RAM. Jest ona ładowana na początku pracy systemu przed przełączeniem na tryb drugi przerwań. Po jej załadowaniu i zapisaniu odpowiednią wartością rejestru I μP może odblokować przerwania. Zastosowanie pamięci RAM pozwala na dynamiczną (w trakcie pracy systemu) zmianę adresów początkowych dla tego samego źródła przerwania. W momencie obsługi przerwania , każdy μP stawia na stosie PC. μP 6502 zapisuje również bajt bitów warunkowych, a 6800 5 bajtów, które razem z PC tworzą pełną kopię jego rejestrów. Jeśli podprogram może zmienić pewne rejestry to na jego początku należy umieścić instrukcję składającą te rejestry na stosie, a na końcu w odwrotnej kolejności instrukcje odtworzeniowe (odczyt ze stosu). Nie dotyczy to 6800. W przypadku Z80 jeśli tylko obsługa przerwań jest jednopoziomowa (przerwanie nie przerywa obsługi przerwania) można wykorzystać alter-

$$\begin{cases} EX & AF, AF' \\ EXX & \end{cases}$$

Rysunek 9: Alternatywny blok rejestrów

natywny blok rejestrów, rys. 2.0.5 (instrukcje str.6 dodatku). Przerwanie umożliwia efektywne wyprowadzenie μP ze stanu zatrzymania (instrukcja HLT dla Z80 i 8080 oraz WAI dla 6800) po wykonaniu takiego rozkazu μP zaprzestaje wykonywania rozkazów następnych. Należy zwrócić uwagę, że 6800 oczekując na przerwanie składa przygotowawczo na stosie cały swój stan. Wyprowadzając μP ze stanu zatrzymania przerwaniem powodujemy, że jako adres powrotny zapisuje on na stosie adres instrukcji następnej za instrukcją zatrzymania, aby nie wracać do stanu zatrzymania ponownie. Jeśli przerwania NMI nie są wykorzystywane, a maskowalne zostały zablokowane to pozostaje jedynie sygnał RESET do wyprowadzenia zatrzymania μP . Zatrzymany μP może oczywiście być wprowadzony w stan zawieszenia i "oddać" magistralę wprowadzając swoje linie w stan wysokiej impedancji.

2.0.6 Porównanie list rozkazów

skoki – Jedynie 6800 dysponuje pełnym asortymentem skoków warunkowych, w których zawarto wszystkie relacje pomiędzy liczbami ze znakiem i bez znaku, a także sprawdzanie stanu pojedynczych bitów warunkowych (C,Z,N,V). Wszystkie te skoki mają adresacje względną (-128, +127), są kodowane dwubajtowo. Instrukcja, która nie uwzględnia warunków (BRA) odpowiada skokowi bezwarunkowemu. Ponieważ zasięg tych instrukcji jest niewielki, więc dla celów przemieszczania programu po całej przestrzeni adresowej zestaw skoków uzłniono instrukcją JMP z pełnym 16- bitowym adresem w trybie absolutnym bądź indeksowym. Przez połączenie dowolnego skoku warunkowego oraz instrukcji JMP zyskujemy możliwość skoku warunkowego w całej przestrzeni. Powyższy zestaw skoków warunkowych stał się niejako standardem. W przypadku Z80 skoki sprawdzają wyłącznie stan pojedynczych bitów warunkowych. Stosuje się adresację bezwzględną i relatywną. Przy bezwzględnej (absolutnej) używa się bitów S,Z,P/V oraz C, a przy adresacji względnej wyłącznie C i Z. Sprawdzanie relacji pomiędzy liczbami ze znakiem wymaga w tym przypadku szeregu skoków, bądź dodatkowych operacji logicznych na bitach warunkowych. Oprócz instrukcji skoków występują tutaj również rozkazy tzw. rozskoków JP(HL), w zależności od tego co znajdowało się w parze rejestrów μP może przenieść bezwarunkowo sterowanie w różne miejsca. W Z80

pojawił się po raz pierwszy rozkaz pomocniczy do organizacji pętli programowej DJNZ kontrolującej ilość jej obiegu. Umieszczenie DJNZ na końcu sekwencji instrukcji powoduje, że będzie ona powtarzana (skok względny przeważnie wstecz), aż do wyczerpania się zawartości rejestru B. W Z80 występują wywołania i powroty warunkowe (warunki identyczne jak przy skokach). Adresacja jest wyłącznie absolutna. Oprócz kodowanej 3-bajtowo instrukcjii CALL występuje również wywołanie 1-bajtowe do ośmiu stałych lokacji tzn. instrukcja restartu RST. W przypadku 6800 występuje wyłącznie odmiana wywołań bezwarunkowych z adresacją względną (BSR) oraz z absolutną lub indeksową (JSR). W obu procesorach na stosie zostaje zapisana zawartość PC. Jest ona odtwarzana instrukcją RST względnie RET z tym, że dla Z80 występują również powroty warunkowe RET CC, RZ, RNZ, RP, RC, RNC, RNP. Oprócz instrukcji RET, która może kończyć także procedurę obsługi przerwania (maskowalnego) występuje także instrukcja RETI oraz RETN. Pierwsza z nich wykona to samo co instrukcja RET aczkolwiek jest inaczej zakodowana. Instrukcja ta jest rozpoznawalna przez układy peryferyjne rodziny Z80 pracujące w łańcuch przerwań Z80. Po zdeklarowaniu tego rozkazu układ, który aktualnie wystawił przerwanie wycofuje je. Rozkaz *RET n* musi być używany jako ostatnia instrukcja podprogramu obsługi NMI, gdyż oprócz odtworzenia PC następuje także odtworzenie stanu zezwolenia na przerwanie maskowalne (wskazuje to IFF1) z przerzutnika IFF2. Wszystkie operacje na bitach (IFF1, IFF2 w tablicach na str. 7).

Przerwania – μP 6800 dysponuje instrukcją przerwania programu **SWI**. Instrukcja ta spełnia wszelkie wymogi odnośnie rozkazów służących do zakładania pułapek programowych (nie jest dłuższa od najkrótszego rozkazu z listy), zachowuje stan na stosie, kieruje μP w inne wybrane miejsce. Podkładając w wybranym miejscu zamiast kodu operacyjnego instrukcję SWI powodujemy, że po dojściu do tego miejsca μP przejdzie do procedury obsługi pułapki (przedstawienie stanu μP na ekranie, ewentualnie jego modyfikacja, zastawianie pułapek w innych miejscach, przywrócenie rozkazu zasłoniętego rozkazem pułapki) SWI po czym RTI. Programową pracę krokową możemy oczywiście zrealizować przy pomocy tzw. kroczącej pułapki. Określenie następnych lokacji rozkazu, który trzeba zasłonić SWI dokonuje się programowo na podstawie analizy zasłoniętego kodu operacyjnego i ewentualnie dalszych bajtów rozkazu oraz w przypadku instrukcji warunkowych na podstawie zawartości bajtu statusowego umieszczonego przez SWI na stosie. Rozkazu analogicznego nie ma na liście Z80. Funkcji tej nie

mogą przejąć rozkazy restartów RST (stałe lokacje). Można zastosować rozwiązanie programowo - sprzętowe, które :

- będzie wykorzystywało komparator adresowy,
- w miejscu pułapki należy wstawić kod instrukcji CALL
- w momencie zrównania się adresów, co stwierdza komparator,należy po odczycie kodu CALL sprzętowo podać na szynę danych jego dwa dalsze bajty określające położenie procedury obsługi pułapki pracy krokowej

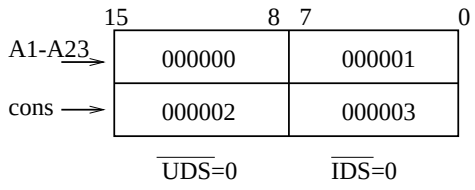
Posiłkując się pułapką wyłącznie sprzętową (komparator wystawiający przerwanie) nie musimy kalkulować adresu następnych instrukcji, gdyż μP złoży ją na stosie po wejściu w obsługę przerwania (czyli pracy krokowej pułapki). Pułapka tego typu jest oczywiście tzw. pułapką " po " , gdyż akceptacja przerwania następuje po dokończeniu instrukcji. Omawiane poprzednio pułapki programowe są oczywiście pułapkami typu "przed". Rozwiązania wykorzystujące przerwania nie mogą być częstokroć stosowane, z drugiej strony pułapka może być założona na dowolny cykl maszynowy, a nie tylko na kod operacyjny jak to ma miejsce przy pułapce programowej.
Charakterystyczne dla Z80 są instrukcje przesłań blokowych. Mogą one być dokonywane pomiędzy pamięcią, a także z udziałem przestrzeni we/wy. Do grupy tej zaliczamy też porównania blokowe. Przed wykonaniem instrukcji blokowej należy oczywiście do odpowiedniej pary rejestrów załadować adres początku bloku źródła, przeznaczenia, bloku porównywanego, ilości przestrzeni czy wzorca do porównywania. Należy zaznaczyć, że po każdym nawrocie procesów powtórnie czyta dwubajtowy kod operacyjny (chodzi tu o to, aby nie zniknęły podczas odświeżania doczepione do cyklu pobrania). Jeśli tylko jest zezwolenie μP może zaakceptować przerwanie po każdym nawrocie instrukcji blokowej po czym wznowić jej realizację po nawrocie z przerwania. Procedura przerywaj musi zapamiętać na stosie i odtworzyć bieżące wartości DE, BC i HL. Odpowiednio na linii BUSRQ μP odda magistralę po każdym dowolnym cyklu maszynowym instrukcji blokowej. W przypadku przesłań blokowych występuje też wersja jednokrotna. Możemy je wykorzystać w celu .....oprócz wyzerowania BC dodatkowego warunku kończącego przesłanie. Podobna sytuacja jest dla przypadku porównań. Porównania blokowe dysponują z *"urzędu"* tylko możliwością sprawdzenia relacji Z=1. Jeśli chcemy wykorzystać inną relacje w celu wyjścia z bloku porównań to użyjemy mutacji CPI bądź CPD. Mimo , iż podaje się , że przestrzeń we/wy Z80 dysponuje 2^8 lokacjami możliwe jest jej rozszerzenie do pełnych 16 - linii adresowych. Wynika to z faktu, że oprócz 8- bitów adresu pojawiającego się na młodszej połowie szyny adresowej μP

wyprowadza jeden z rejestrów na połowę górną. Tak więc zawartość tego rejestru umożliwia wspomnianą powyżej ekspansją przestrzeni. Jest ona najbardziej efektywna za wyjątkiem instrukcji, której adres w przestrzeni we/wy podaje się bezpośrednio (n). Oprócz przesłań pojedynczych możliwe są również przesłania blokowe pomiędzy układem we.lub wyj.,a pamięcią. Maksymalna długość bloku wynosi tutaj 2^8. Oprócz wersji ze zwiększaniem i zmniejszaniem HL-a , są też mutacje jednokrotne w których można zrealizować własny warunek zakończenia transmisji (procesor powtarza, aż do wyczerpania).

3 MIKROPROCESOR 68000

Układ ten dał początek całej rodzinie opartej na ustandaryzowanej architekturze i ujednoliconym zestawie instrukcji - procedury układu poprzedniego są wykonywane również przez μP nowszych typów.
Do rodziny tej zaliczają się układy[2]:

- 68000 / 68008
- 68010
- 68020
- 68030
- 68050
- 68060



Rysunek 10:

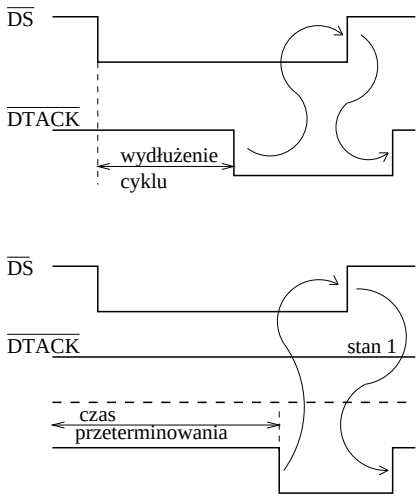
Wszystkie μP z rodziny posiadają wewnętrzną architekturę 32-bitową. μP może pracować w trybie nadzorcy , gdzie dysponuje pełnymi zasobami w trybie użytkownika, gdzie dostęp do pewnych bloków oraz istotnych instrukcji jest niemożliwy. W sensie zewnętrznym 32- bitowa szyna adresowa i danych pojawiła się dopiero od układu '20. Poprzednie μP posiłkowały się 16-bitową szyną danych, a przestrzeń adresową miały sztucznie skróconą do 24 linii. Architektura wewnętrzna składa się dla pracy na poziomie użytkownika we wszystkich μP rodziny z 8-rejestrów danych i 8- rejestrów adresowych. Rejestry te są całkowicie symetryczne pod względem realizowanych funkcji. Ostatni z 7- rejestrów adresowych jest zdwojony i ma własności domyślnego wskaźnika stosu oddzielnego dla poziomu nadzorcy i użytkownika.

Wprowadzenie ujednoliconego formatu instrukcji kodu operacyjnego zajmują zawsze dwa bajty podobnie jak adresy czy dane, rys. 10. μP może pobrać instrukcję tylko wtedy jeśli zaczyna się ona od adresu parzystego.Te tzw. wyrównanie binarne powoduje przyspieszenie pracy μP . Nie jest ono wymagane (ale jest zalecane) także przy operacjach na danych. Oprócz rozdziału zasobów (ograniczenie na poziomie użytkownika) μP separuje sprzętowo pamięć dla obu tych poziomów wprowadzając rozróżnienie także na program i dane. O tym, która 2^{24} bajtowa przestrzeń jest używana informuje linia FC0-FC02 będąca tak jakby rozszerzeniem szyny adresowej. Podobnie jak inne μP firmy Motorola 68000 nie posiada wydzielonej przestrzeni we/wy - układy te są lokowane w przestrzeni pamięciowej. Ponieważ μP

[2]Wojciech Czyż *Rodzina M680xx*

68000 posiada kolejkę rozkazów do wykonania więc operacje zewnętrzne nie nazywamy już cyklami maszynowymi lecz cyklami magistrali.

W ogólnym przypadku ilość cykli magistrali jest zawsze większa od ich niezbędnej ilości z tym, że μP wyposażony w kolejkę będzie zawsze szybszy od konstrukcji klasycznej. μP stosuje asynchroniczny cykl magistrali (możliwość wstawienia taktów oczekiwania) i jest przystosowany sprzętowo do pracy w systemie normalnie niegotowym. Każda karta systemu po zaadresowaniu musi ingerować w linię DTACK sprowadzając ją na niski aktywny stan natychmiast, gdy nie wymaga taktów oczekiwania, bądź z pewnym opóźnieniem w przypadku wolnych komponentów systemu(rys. 11).
Tak więc wystawienie przez μP adresu nieobsadzonego spowoduje, że nie otrzyma on potwierdzenia i zacznie wstawiać bez końca takty oczekiwania. Zapobiega temu zewnętrzna logika przeterminowania. Odmierza ona od początku każdego cyklu czas dłuższy od najwolniejszej karty, po czym jeśli do tej pory nie pojawi się potwierdzenie wysterowuje BERR, μP kończy wtedy arbitralnie cykl i rozpoczyna tzw. obsługę błędu magistrali. Podobna sytuacja może wystąpić w przypadku uszkodzenia sprzętowego jednego z komponentów systemu.



Rysunek 11: Logika potwierdzenia $\overline{DTACK}$

Ze względu na możliwość współpracy z układami 6800 wprowadzono możliwość generacji cyklu synchronicznego. Fakt ten ma miejsce jeśli karta zawierająca takie układy na podstawie rozpoznanego własnego adresu wysteruje linię VPA. W generowanym wówczas przez μP cyklu synchronicznym pojawiają się sygnały właściwe dla 6800, a to E(02), VMA, R/W. Funkcja linii adresowych i danych jest oczywiście identyczna jak w cyklu asynchronicznym. Cykl synchroniczny nie sprawdza stanu DTACK i jest odpowiednio wolniejszy (linia E jest podzielona przez 10 zegarem CLK).

Cykl synchroniczny wykorzystano ponadto dla sygnalizacji tzw. autowektoryzacji przy obsłudze przerwań. Przy stwierdzeniu błędu magistrali istnieje możliwość wielokrotnego powtarzania cyklu, dzieje się tak jeśli równocześnie z linią BERR wysteruje linię HALT. Linia HALT staje się aktywną linią wyjścia w przypadku tzw. podwójnego błędu, który uniemożliwia dalszą pracę μP . Wprowadza ona swoje linie w stan nieaktywny bądź trzeci (wy-

prowadzenie z tego stanu to podanie sygnału RESET). Przy włączonym zasilaniu używa się jak wspomniano sygnału RESET, natomiast po włączeniu napięcia zasilającego pełne wyzerowanie μP następuje po równoczesnym podaniu sygnału RESET i HALT. Linia RESET jest linią dwukierunkową , ustawia się na wyjście podczas realizacji rozkazu RESET (dostępny wyłącznie na poziomie nadzorcy). Podobnie dwukierunkowa jest linia HALT. Wykorzystując wyłącznie tę linię jako wejście uzyskujemy możliwość pracy co jeden cykl maszynowy, z tym że zatrzymanie pracy μP następuje po dokończeniu cyklu tak, że nie jest możliwa obserwacja stanu linii co może mieć np. miejsce jeśli np. do krokowania wykorzystana zostanie linia DTACK (logika przeterminowania wyłączona).

NMI - jest zawsze akceptowane zboczem

Dotychczas poznane μP dysponowały zwykle 3 poziomami przerwań:

- brak
- $\overline{INT}$
- $\overline{NMI}$

W 68000 występuje 8 poziomów:

- brak
- $\text{"rozszczepione"} INT = \begin{cases} INTR1 \\ \vdots \\ INTR6 \end{cases}$
- INT7 - NMI (aktywne zboczem)

Linie IPL0-2 nie są wyprowadzone na magistralę, gdyż mogłoby dojść do zsumowania poziomów. Na magistralę wyprowadza się 7 linii żądań INTR1-7, z których na karcie CPU przy pomocy enkodera priorytetów '148 odtwarza się stan linii IPL. Przypadek NMI następuje jeśli na co najmniej jednej linii IPL wystąpi zbocze opadające, a pozostałe będą w aktywnym niskim stanie. μP może oddać magistralę po każdym cyklu magistrali, logika jest tutaj inna niż w poznanych do tej pory procesorach. W odpowiedzi na żądanie BR μP natychmiast zgłasza chęć oddania magistrali na linii BG. Nie musi to wcale oznaczać, że została ona już zwolniona. Będzie ona zwolniona dopiero jeśli μP zakończy bieżący cykl. Fakt zakończenia cyklu musi rozpoznać sam żądający. Dokonuje on tego na podstawie obserwacji linii AS

i DTACK (obie w nieaktywnym wysokim poziomie). Dopiero w tym momencie można zająć magistralę, a do μP wysyła się sygnał BGACK informujący go. Linia BGACK przeważnie odłącza też bufory pośredniczące między μP , a magistralą. Wysterowanie BGACK powoduje też, że dalsze zgłaszanie żądania BR może zostać wycofane.
W stosunku do 68000 istnieje jego mutacja 68008 przystosowana do pracy z 8 - bitową szyną danych. W związku z tym pojawiła się linia A0, a zamiast strobów UDS i LDS wspólny strob DS. Przestrzeń adresowa została jeszcze bardziej zredukowana (1Mb) w stosunku do 16 Mb 68000 , ponadto wskutek zwarcia wewnętrznego IPL0 i IPL2 μP akceptuje poziomy 0,2,5 i 7. Wyeliminowana została linia BGACK, tak więc linii BR nie można przedwcześnie wycofać. Oprócz tego założono, że sygnał VMA będzie wytwarzany zewnętrznie. μP korzysta z pierwszych dwóch wektorów w tabeli stanów wyjątkowych zapisując wartościami spod adresu 0-3 SSP(stos nadzorcy), a 4-7 PC. Cała tabela stanów wyjątkowych znajduje się w obszarze danych nadzorcy i jest ulokowana w pamięci RAM. Jedynie dwa pierwsze wektory (nakładkowe) inicjowane po zerowaniu są odczytywane z obszaru kodu nadzorcy , w miejscu tym musi znaleźć się pamięć stała. Pierwsze 64 lokacje zarezerwowane są dla procesora, pozostałe 64-256 są przeznaczone dla użytkownika. W przypadku zgłoszenia przerwania μP porównuje stan linii IPLK z bitami maski w słowie statusowym (bajt systemowy). Jeśli zgłaszany poziom jest większy od poziomu maski to możliwa jest akceptacja tego zgłoszenia, polega ono na tym, że po zapisaniu na stosie wartości PC i rejestru statusowego, μP wykona cykl akceptacji przerwania. Cykl ten można rozpoznać po tym, że wszystkie linie FC są w wysokim poziomie. W cyklu tym μP oczekuje, że po liniach D0-D7 zostanie mu przesłany numer wektora. Numer ten określi w tablicy wektorów stanów wyjątkowych związany z nim adres z pod którego rozpocznie się obsługa przerwania (wszystkie adresy w tabeli są oczywiście parzyste, za wyjątkiem nakładkowej wartości SSP). W cyklu akceptacji przerwania μP jedynkuje wszystkie linie adresowe za wyjątkiem A1-A3. Na tych trzech liniach podaje nr akceptowanego aktualnie poziomu. Dlatego też urządzenie zgłaszające przerwanie musi zaczekać na cykl akceptacji na swoim poziomie (jeśli jest więcej urządzeń na danym poziomie muszą dojść do porozumienia). Po wejściu do obsługi przerwania μP "podnosi" stan bitów **I0 i I2**(w bajcie systemowym) do poziomu zaakceptowanego. Tak więc na poziomie N nie może przerwać samego siebie. Od warunków wyższości IPL nad bitami maski przerwań istnieje tylko jeden wyjątek związany z NMI (poziom 7). Będzie on akceptowany nawet wtedy, gdy na bitach I0 do I2 znajduje się 1. Zapisując na stosie przed wejściem w obsługę bajt systemowy μP zachowuje tam poziom I0 do I2 sprzed akceptacji przerwania. Poziom ten jest przywracany instrukcją

RTE kończącą obsługę przerwania. Obsługa przerwania i każdego innego stanu wyjątkowego dokonywana jest zawsze na poziomie nadzorcy (bit S zostaje ustawiony, a bit T wyzerowany). Bit T w słowie statusowym jeśli jest ustawiony powoduje, że μP zgłasza przerwanie wewnętrzne od tzw. śledzenia po wykonaniu każdego rozkazu. Następuje wówczas zapis na stosie statusu i PC, po czym do PC ładowane są 4 bajty wpisane pod adresami 000024-27H. Adresy te zawierają adres od, którego rozpoczyna się obsługa pracy krokowej (μP przekazuje np. na monitor stan swoich rejestrów). Obsługa pracy krokowej kończy się tak jak i każdy stan wyjątkowy instrukcją RTE. Po odtworzeniu PC i statusu μP wykona kolejny rozkaz po czym znów przejdzie do procedury śledzenia. Wyłącznie dla nadzorcy zastrzeżone są następujące instrukcje RESET, RTE, STOP oraz wszelkie sposoby zmiany bajtu systemowego (ładowanie, suma logiczna, iloczyn logiczny, EXOR). RTE jest zaliczana do instrukcji uprzywilejowanych, gdyż również modyfikuje bajt systemowy zczytując go ze stosu. Widzimy więc, że użytkownik nie mając dostępu do bajtu systemowego nie może ustawić bitu S, T ani zmienić poziomu akceptacji przerwania. To jeszcze nie wszystko, gdyż instrukcją Uprzywilejowaną jest rozkaz **MOVE**, oznacza to, że użytkownik korzystał ze stosu, którego początek ustawił mu nadzorca. Dostęp użytkownika do stosu nadzorcy jest oczywiście niemożliwy z definicji jako, że zależnie od stanu bitu S procesor przełancza rejestry A7 będące wskaźnikami stosu dla obu poziomów. Użytkownik ma tylko jedną legalną możliwość przejścia do poziomu nadzorcy, są to instrukcje TRAP0 do TRAP15. Przy ich pomocy następuje tak jakby wywołanie pewnych procedur zasobów nadzorcy. Procedury takie kończą się oczywiście zawsze instrukcją RTE, która wymusza przejście na mniej uprzywilejowany poziom. Dostęp do danych nadzorcy jest możliwy o tyle o ile zezwalają na to instrukcje składające się na procedury TRAP'-ów. Istnieje wprawdzie 16 instrukcji TRAP, lecz można je oczywiście wywoływać z parametrem w jednym z rejestrów danych. Dzięki temu następuje rozszczepienie na odpowiednią ilość procedur. W normalnej sytuacji w przypadku przerwania do μP dosyłany jest jeden z wektorów o nr 64-256. μP pozwala oprócz tego na rozróżnienie pewnych szczególnych sytuacji, jest to autowektoryzacja, przerwanie fałszywe i niezaprogramowany wektor przerwania.
Ponieważ w celu wystawienia wektora niezbędna jest logika i trójstanowy bufor, głównie w prostych systemach o małej liczbie przerwań można wykorzystać autowektoryzację, w tym celu urządzenie przerywające jeśli stwierdzi cykl akceptacji swojego poziomu wysterowuje na aktywny niski poziom linię VPA. μP przejdzie do cyklu synchronicznego (stan DTACK ignorowany). W tym przypadku oznaczać to będzie, że μP wykorzysta dla każdego poziomu odpowiedni wektor ze stałej lokacji tabeli stanów wy-

jątkowych. Przesłanie wektora od urządzenia przerywającego musi zostać potwierdzone przezeń na linii DTACK. Jeżeli potwierdzenia nie będzie to włączy się logika przeterminowania z wysterowaniem BERR. W tym konkretnym przypadku nie będzie to jednak oznaczało błędu magistrali, lecz przypadek fałszywego przerwania. Ponadto fałszywe przerwanie stwierdzi μP jeśli przejdzie do cyklu akceptacji, a nie zastanie już zgłoszenia na akceptowanym, poziomie (urządzenie się "rozmyśliło"). Układy peryferyjne systemu mają programowany (wpisywany przez μP) nr wektora pod którym zgłaszają się w cyklu akceptacji. Jeśli nr. ten przed cyklem akceptacji nie będzie zapisany przez μP to urządzenie takie będzie posiłkować się zastępczym wektorem nr 15. Odbiór takiego wektora oznacza konieczność zaprogramowania urządzeń, a także rozpoznania urządzenia przerywającego (np. drogą odczytu ich rejestrów statusowych).

3.1 Stany wyjątkowe

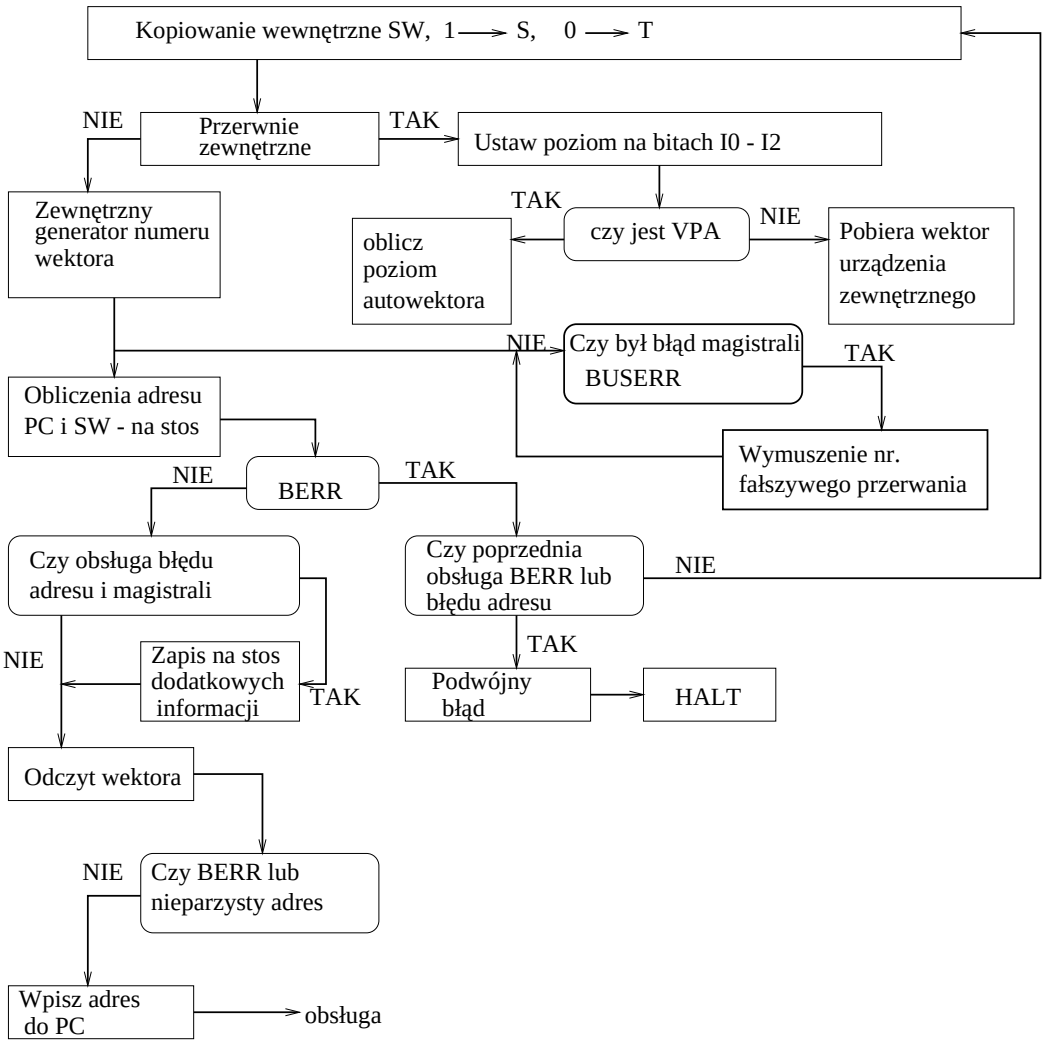
GRUPA		Warunek akceptacji	Na stosie adres rozkazu	↓ malejący priorytet
A	Reset			
	BERR ADR ERR	koniec cyklu zegara	"ramka stosu"	
B	TRACE, INT	koniec rozkazu	następnego	
	emul A,F; naruszenie uprzywilejowania	koniec cyklu magistrali	bieżącego	nie występuje równocześnie
C	TRAP, TRAPV CHK, T0	Koniec rozkazu	następnego	

Tabela 6: Tablica stanów wyjątkowych

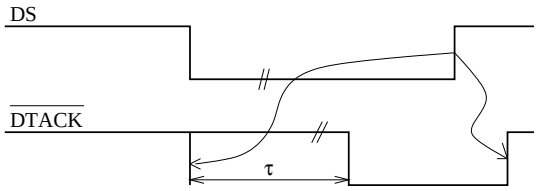
Stany wyjątkowe (tab. 6) pod względem hierarchi możemy podzielić na trzy grupy (0,1,2,3). Do grupy zerowej zaliczamy zerowanie μP , do grupy 1 - błędy adresu i magistrali, a do grupy 2 - pozostałe stany wyjątkowe.
W grupie 0 - przejście do zerowania jest możliwe po każdym takcie zegara, w grupie 1 po każdym cyklu magistrali, a w grupie 2 dopiero po zakończeniu instrukcji. Grupa 2 składa na stosie PC oraz rejestr statusowy, natomiast w grupie 1 oprócz powyższych 6-bajtów μP składa na stosie dodatkowo 8-bajtów, tworząc tzw. ramkę stosu. Są to dodatkowe informacje pozwalające na podjęcie bardziej skutecznej obsługi przez odnośne procedury.

3.1.1 Sieć działań obsługi stanów wyjątkowych

Obsługa każdego stanu wyjątkowego (rys. 12 dokonywane jest na poziomie nadzorcy. Na czas obsługi stanu wyjątkowego zostaje zawieszone śledzenie. Rozkaz RTE kończący obsługę, każdego stanu wyjątkowego przywraca poprzednią wartość słowa statusowego (np. powtórnie inicjuje śledzenie), przywraca poprzedni poziom akceptacji przerwania, kieruje μP np. z powrotem na poziom użytkownika.
Powyższy graf przedstawia kolejne działania μP podczas obsługi jakiegokolwiek stanu wyjątkowego. Przez obsługę rozumiemy tutaj wyłącznie proces wejścia do programu obsługi. Składa się on z operacji stosowych, ewentualnego cyklu akceptacji przerwania i z operacji odczytu wektora z tabeli stanów wyjątkowych.
Podczas tych czynności μP może nie uzyskać potwierdzenia na linii DTACK lub natrafić na nieparzysty adres. Brako potwierdzenia i w konsekwencji stwierdzenie błędu magistrali nie ma miejsca jedynie w cyklu akceptacji przerwania. Nieparzysty adres natomiast może wystąpić jedynie podczas odczytu z tabeli stanów wyjątkowych nieparzystej wartości PC. Jeśli w danej chwili μP rozpoczynał wejście w obsługę stanów wyjątkowych od numeru 4 lub rozpoczynał obsługiwać przerwanie to powyższe błędy magistrali lub adresu przerwą te czynności i skierują μP na początek sieci działań, gdzie rozpocznie się obsługa wejścia w głąb magistrali lub adresu. Jeśli natomiast błędy adresu bądź magistrali wystąpiły już podczas obsługi stanów wyjątkowych błedu magistrali lub adresu to μP stwierdza podwójny błąd i zawiesza się wysterowując na zewnątrz linię HALT. Należy zauważyć, że zawieszenie μP wystąpi przy błędzie pojedynczym jeżeli w trakcie ładowania wartości początkowej do SSP i PC zostanie stwierdzony błąd magistrali lub do PC wpisany zostanie adres nieparzysty.
Nieparzysta wartość SSP, aczkolwiek dopuszczalna nie jest zalecana ze względu na spowolnienie szybkości działania μP .
μP 68000 koduje rozkazy na 16-bitach, w kodzie operacyjnym mogą wystąpić 6-bitowe pola określające adres efektywny (jedno bądź dwa).
Dla niektórych rozkazów pewne tryby adresacji nie mają sensu podobnie jak nieodpowiednia wartość w dwubitowym polu rozmiaru argumentu. Jeśli rozkaz taki (niedozwolonym trybem adresacji bądź rozmiarem) zostanie pobrany to wystąpi stan wyjątkowy nielegalnej instrukcji. Oprócz tego stan ten będzie zainicjowany dla trzech zarezerwowanych kombinacji kodowych. Wyjątek dzielenia przez zero jest inicjowany zawsze jeśli taka sytuacja wystąpiła. Należy zauważyć, że ustawienie bitu nadmiaru (V) nie



Rysunek 12: Algorytm obsługi stanów wyjątkowych



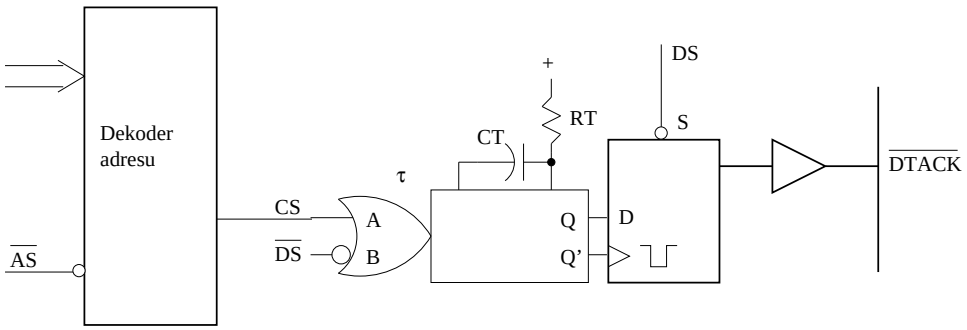
Rysunek 13: Logika potwierdzenia

inicjuje automatycznie wyjątku nr.7. Jego inicjacja następuje tylko wtedy, jeśli μP odczyta rozkaz TRAPV przy ustawionym bicie nadmiaru. Próba wykonania instrukcji uprzywilejowanej przez użytkownika to oczywiście stan wyjątkowy narzucenia uprzywilejowania. Jeśli na poziomie nadzorcy zostanie ustawiony bit T to μP po każdym rozkazie będzie inicjował automatycznie wyjątek śledzenia. Kody operacyjne rozkazów rozpoczynające się od A i F nie są w 68000 wykorzystywane. Próba podania takich kodów spowoduje wyjątek 10 lub 11. Tzw. emulator linii F jest przeznaczony dla programowej symulacji rozkazów kooprocesora, jeśli nie jes on dołączony. Począwszy od 68040 wyjątek ten nie występuje w związku z zainstalowniem w μP logiki kooprocesora.
Emulator linii A jest przeznaczony do wykorzystania przez użytkownika. Przy pomocy tego wyjątku uzupełnionego np. parametryzacją przez jeden z rejestrów możemy zdefiniować swoje własne instrukcje emulowane programowo. Stany wyjątkowe, które kładą na stosie swój niepowiększony stan PC (nielegalny, emulatory, naruszenie) wymagają przy swojej obsłudze modyfikacje tej wartości na stosie, aby rozkaz RTE kończący ich obsługę nie skierował μP z powrotem do tego samego miejsca.
W systemie 68000, każda karta winna być wyposażona w logikę potwierdzenia transmisji ($\overline{DTACK}$, rys. 13), jeżeli nawet nie wymaga ona dodatkowych taktów oczekiwania (rys. 14).

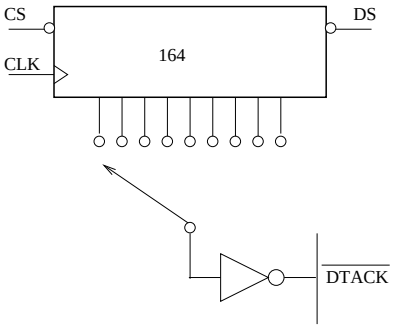
Powyższe przykłady realizacji logiki DTACK wykorzystują strob danych (rys. 15, a nie adresu. Jest to konieczne ponieważ μP może też wykonać cykl-odczyt-modyfikacja-zapis (1-AS, 2-DS).

Stosowanie linii AS w powyższych przykładach spowodowałoby przeterminowanie i błędy magistrali. Ponieważ odmierzanie czasu przeterminowania następuje zewnętrznie, więc na karcie μP muszą znaleźć się odnośne układy sprzętowe (rys. 16).

Oprócz krokowania programowego (ustawiony bit T) można stosować krokowanie sprzętowe co cykl magistrali. Do tego celu możemy wykorzy-



Rysunek 14: Dodatkowe takty oczekiwania dla linii $\overline{DTACK}$

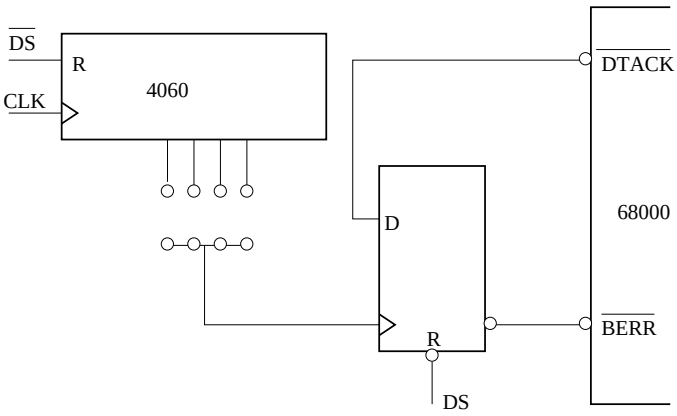


Rysunek 15: Przykład realizacji opóźnienia

stać linię HALT bądź linię DTACK. Krokowo z użyciem linii HALT nie pozwala na bezpośrednią obserwację stanu magistrali. (μP kończy cykl i wchodzi w stan wysokiej impedancji).
Bardziej poglądowe krokowanie (rys. 17 uzyskujemy wykorzystując linię DTACK z tym, że nie możemy zaburzyć potwierdzeń od modułu systemu oraz nie możemy prowokować błędu magistrali. Rozwiązaniem tego problemu jest zdefiniowanie dodatkowej linii $\overline{DTACK_c}$.

3.2 Wybrane zagadnienia listy rozkazów

Wśród bitów warunkowych w bajcie użytkownika pojawił się bit **X**. Z bitu tego korzystają bądź ustawiają go instrukcje w których mnemonik zakończony jest literą **X**.
Znaczenie bitu X jest identyczne jak przeniesienie CARRY. Bitu X używa się przy operacjach arytmetycznych wielokrotnej precyzji. Dzięki bitowi X związane z tą arytmetyką instrukcje nie natrafiają na zniszczoną wartość



Rysunek 16: Przeterminowanie z wykorzystaniem linii $\overline{BERR}$

przeniesienia, które międzyczasie może zmienić inny rozkaz (np. testujący warunek końca obiegu pętli).
W instrukcjach 68000 może wystąpić jedno bądź dwa 6-bitowe pola precyzujące tryb adresacji argumentu (argumentów). Należy zaznaczyć , że nie wszystkie tryby są logicznie uzasadnione, przy każdym rozkazie wpisanie nieodpowiedniego trybu przy danej instrukcji spowoduje inicjacje stanu wyjątkowego nielegalnej instrukcji. W 6-bitowym polu 3 bity przeznaczone są na zdefiniowanie trybu , a pozostałe 3 adresują rejestr (adresowy bądź danych) bądź stanowią rozszerzenie trybu (rys. 7.

TRYB	Rejestr		
000	xxx	Dn	-arg. jest rej. danych
001	xxx	An	-arg. jest rej. adresowy
010	xxx	(An)	- adr. pośrednie
011	xxx	(An)+	-postincrementowe
100	xxx	-(An)	-predecrementowy
110	xxx	b(An,Dn,L)	pośrednie
		b(An,Dn,W)	z indeksem,
		b(An,Am,L)	wymaga tzw.
		b(An,Am,W)	słowa rozszerzenia
101	xxx	W(An)	pośrednie z przesunięciem (wartość W za KO)
111	000	W	absolutne krótkie
111	001	L	absolutne długie

111	010	W(PC)	względne
111	011	b(PC,Dn,L)	względne
		b(PC,Dn,W)	z indeksem
		b(PC,An,L)	(wymaga słowa
		b(PC,An,W)	rozszerzenia)
111	100	X	natychmiastowe (dotyczy tylko "źródła")
		SR, CCR	adresowanie rejestru dotyczy tylko "przeznaczenia"

Tabela 7: Tryby adresacji

A	rejestr	L	000	8-bitowe przesunięcie

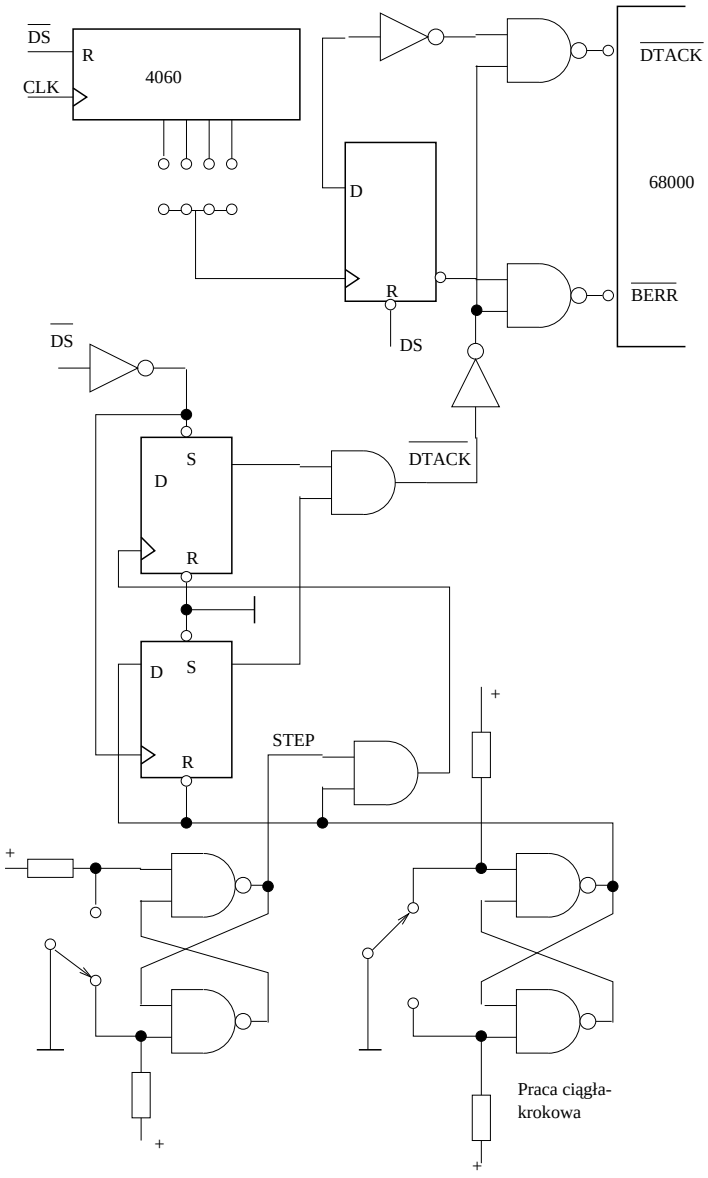
Tabela 8: Słowo rozszerzenia

- A = 1 dla Am
- A = 0 dla Dn

L = 1 rozmiar długości słowa (32 - bity) L

L = 0 rozmiar słowa (16 - bity) W

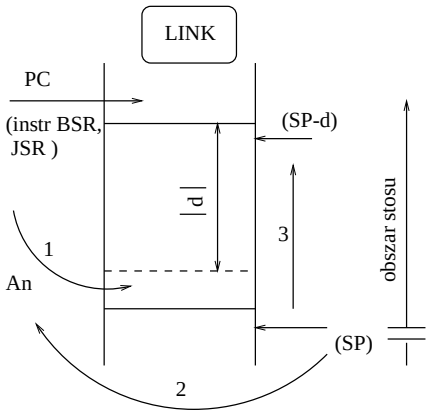
Za wyjątkiem rozkazu **PEA** na liście nie występują w sposób jawny instrukcje zapisu na stos i odczytu ze stosu. Kontakty ze strukturą danych będącą stosem są tutaj jednym z przypadków adresacji. Przy zapisie na stos stosuje się tryb predecrementowy , a przy odczycie postincrementowy. Jako wskaźnika stosu można użyć każdy z rejestrów adresowych. Wskaźnik stosu z "urzędu", którym jest rejestr A7 (A7') μP wykorzystuje oczywiście przy instrukcjach wywołań powrotów , a także przy rozkazach LINK i UNLK. Często stosowaną metodą przekazywania parametrów wejściowych dla wywoływanego z różnych miejsc podprogramu jest sposób przekazywania ich za pomocą stosu. W tym celu należy przed wywołaniem podprogramu wpisać na wierzchołek stosu wspomniane parametry, a dopiero potem wywołać podprogram. Podprogram ten po wykonaniu instrukcji CALL położy na stosie PC oraz najprawdopodobniej wartość innych rejestrów. Po wykonaniu swoich czynności wystąpi odtworzenie stanu μP , a rozkaz powrotu przywróci poprzednią wartość PC. Jednak każdorazowo taka operacja spowoduje narastanie stosu, który może dojść do innych danych i zdezorganizować pracę systemu. Stosując taką metodę należy więc po każdorazowym



Rysunek 17: Przykład realizacji pracy krokowej

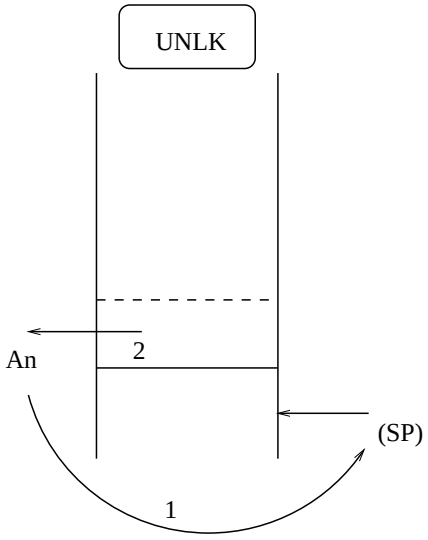
wywołaniu wspomnianej procedury obniżyć (zdecrementować) wskaźnik SP. Problemy te rozwiązują całościowo instrukcje **LINK i UNLK**.

Program główny wykonuje wpierw instrukcję LINK potem zapisuje parametry do podprogramu, a następnie wywołuje i sam podprogram (rys. 18. Aby utworzyć *lukę* na stosie należy zmodyfikować (przeważnie zawsze zmniejszyć) wartość SP. Ponieważ SP niemodyfikowane będzie zawsze potrzebne po powrocie do podprogramu głównego dlatego jego wartość przechowujemy w jednym z jego rejestrów adresowych, aby zaś nie stracić rejestru **An** musimy zachować go na stosie. Tak więc powyższe operacje wykonuje μP oczywiście w odwrotnej kolejności. Przesunięcie "d" jest 16-bitowe i zajmuje następne słowo za KO. Program główny wpisując parametry dla procedury, a także podprogram korzystając z nich może używać następujących trybów adresacji:



Rysunek 18: Działanie instrukcji LINK

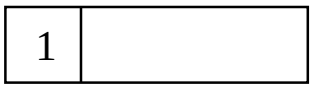
- W(An)
- -(An)
- (An)+



Rysunek 19: Działanie instrukcji UNLK

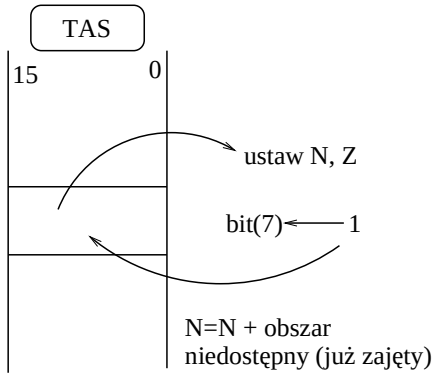
Podprogram oczywiście na samym początku może jeszcze zapamiętać na stosie inne rejestry, a przed powrotem je odtworzyć. Po powrocie z procedury program główny może wykorzystać zarezerwowany obszar stosu, w którym przykładowo procedura umieściła swoje parametry wyjściowe, lecz potem luka na stosie musi zostać zlikwidowana. Dokonuje tego (w programie głównym) instrukcja **UNLK**(rys. 19). Jeśli zachodzi konieczność zachowania na stosie (bądź, gdzie indziej) stanu μP używamy instrukcji **MOVEM**. Jej dwa rodzaje powodują zachowanie lub odtworzenie maksymalnie wszystkich rejestrów adresowych i danych, obszar którego początek wskazuje adres efektywny, w KO

w drugim słowie rozkazu w formie zero - jedynkowej zaznacza się, które rejestry będą zapisywane (odtworzone), a które nie. W przypadku pracy wielo-μP , poszczególne procesory muszą dzielić zasoby pamięciowe i układy we/wy pomiędzy siebie. Chodzi o to, żeby przy cyklicznym aktywowaniu poszczególnych μP przez arbiter, kolejny aktywny μP nie zaczął korzystać z obszaru pamięci, z którą ostatnio poprzednio pracował inny μP , którego działanie zostało zawieszone przez arbiter. W tym celu odpowiednim blokom pamięci bądź układom we/wy przyporządkowujemy bajty w pamięci, gdzie znajdują się związane z powyższymi zasobami tzw. semafory.



Rysunek 20: Semafor

W przypadku 68000 dany semafor wskazuje, że z przyporządkowanego mu obszaru korzysta już jakiś μP jeśli na najstarszym bicie znajduje się 1 – rys. 20. Dany μP , który pragnie skorzystać z pewnych zasobów musi najpierw odczytać odpowiedni semafor, sprawdzić czy nie wskazuje on zajętego już obszaru poczym winien umieścić 1 na najstarszym bicie semafora. Gdyby czynności te (odczyt zapis) były podzielone (zawieszenie μP pomiędzy odczytem a zapisem) to w trakcie operacji na semaforze inny μP , któremu arbiter właśnie przekazał magistralę mógłby również rościć pretensję do tego samego obszaru. Kolizją takim zapobiega niepodzielność instrukcji operujących na semaforach **TAS** – rys. 21.



Rysunek 21: Operowanie na semaforach

3.2.1 Rozkazy skoków

μP dysponuje dwoma typami instrukcji przemieszczającymi w inne miejsce pamięci programu, mianowicie rozkaz **Bcc** oraz **JMP**. W przypadku tej pierwszej jedynym dopuszczalnym trybem adresacji jest tryb względny

0111	kod względny	8-bitowe przesunięcie
16-bitowe przesunięcie (ważne, gdy 8-bitowe=0...0)		

Rysunek 22: Instrukcja Bcc

w stosunku do PC. Instrukcja dysponuje 4-bitowym polem określającym jaki warunek musi być spełniony, aby skok został wykonany. Jeden z tych warunków zdefiniowany jest jako "żaden". Oznacza on podobnie zresztą jak i dla JMP skok bezwarunkowy. Dla instrukcji BCC jest możliwy zasięg $\pm 2^7$, bądź $\pm 2^{15}$ – rys. 22.

Jak widać rozkaz Bcc nie umożliwia przemieszczania się po całej przestrzeni adresowej. Możliwość taką ma instrukcja skoku bezwarunkowego JMP, z tym że nie przewiduje się trybu adresacji w formie bezpośredniej danej za kodem operacyjnym.
Wśród dopuszczalnych trybów adresacji występują:

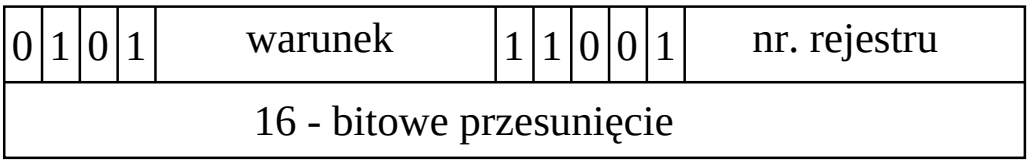
- (An)
- (An)+
- -(An)
- x(An)
- x(An Dm)

Do instrukcji skoków zalicza się również rozkaz organizujący pętle programu **DBcc**. Instrukcja ta definiuje jeden z rejestrów D jako 16-bitowy licznik za każdym razem dekrementowany. Oprócz tego występuje 4-bitowe pole definiujące również 1 z 16 warunków. Tak więc μP może wyjść z pętli, albo gdy licznik w rejestrze D osiągnie wartość zerową lub gdy przy wykonaniu tej instrukcji zostanie stwierdzony zdefiniowany warunek. Wśród tych 16 warunków jeden dotyczy nie brania pod uwagę żadnych warunków, a drugi z kodów cc powoduje, że pętla nie będzie ani razu powtórzona.

3.2.2 Rozkazy wywołań

Rozkaz DB dysponuje zawsze zasięgiem +/- 32 kB (adresacja względna) – rys. 23.

W przypadku wywołań dysponujemy podobnie jak przy skokach instrukcjami **BSR i JSR**. W przypadku BSR podobnie rozwiązano problem 8 bądź 16 bitów kodu przesunięcia bezwzględnego. Jeśli zachodzi konieczność wywołania na dalszy dystans używamy rozkazu JSR. Należy zwrócić



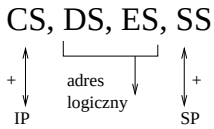
Rysunek 23: Instrukcja DB

uwagę, że dla BSR nie występuje pole określające warunek, oznacza to, że μP posiada wyłącznie rozkazy wywołań bezwarunkowych. Jeśli zachodzi konieczność wywołania podprogramu dla pewnych warunków to należy połączyć działanie instrukcji Bcc i BSR względnie JSR. Zarówno BSR jak JSR muszą oczywiście umieszczać na stosie adres następnej instrukcji dla prawidłowego zadziałania rozkazu powrotu kończącego podprogram. Rozkazem tym jest instrukcja RTS. Instrukcja RTS podobnie zresztą jak inne powroty operuje w całym obszarze adresowym. Oprócz RTS występują również powroty **RTE** i **RTR**. Ten pierwszy służy wyłącznie do zakończenia procedury obsługi stanu wyjątkowego i może być użyty wyłącznie na poziomie nadzorcy, to jest stanu gdzie odbywa się obsługa stanów wyjątkowych. Instrukcja RTR oprócz PC odtwarza również kod bajtów warunkowych nie modyfikując bajtu systemowego.
Wśród operacji bitowych w stosunku do poznanych już instrukcji pojawił się rozkaz **BCHG.** W tym przypadku wskazany bit oprócz przeładowania go do flagi **Z** zostanie ustawiony na swoim miejscu w stanie przeciwnym. W celu zapamiętania w wybranym miejscu informacji o spełnionym bądź niespełnionym warunku wprowadzono instrukcje **SCC**, ustawia ona bajt wskazany przez tryb adresacji w stan jedynek, bądź zer w zależności od tego czy wskazany warunek był spełniony czy nie. Dla wygody pisania procedur relokowalnych ich wznawiania i zawieszania wprowadzono rozkazy **LEA i PEA.** Rozkazy te na podstawie 6-bitowego pola wyznaczają adres efektywny, lecz z niego nie korzystają. Adres ten jest zapamiętywany w pamięci bądź na stosie w celu późniejszego wykorzystania.
Na liście instrukcji w sposób jawny nie występują rozkazy ($\pm$1), zamiast nich mamy do dyspozycji rozkazy **ADDQ i SUBQ.** Odpowiadają one odpowiednio zwiększaniu bądź zmniejszaniu argumentu wskazanego przez adres efektywny o wartość z przedziału 1 do 8, tak więc instrukcje DEC bądź INC poznane wcześniej są jednym z przypadków tych rozkazów. Oprócz tego dla szybkiego ładowania bajtu w miejsce wskazane adresem efektywnym używamy instrukcji **MOVQ** (bajt w kodzie operacyjnym).

4 MIKROPROCESOR 8086

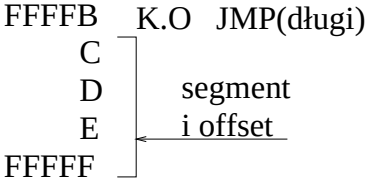
4.1 Krótki opis architektury

Układ 8086[3] dał początek następnym μP Intela. Zapewniono przenoszalność kodu do nowszych typów, a wprowadzając w nich tzw. tryb wirtualny z ochroną zasobów zachowano działanie 8086 w ten sposób, że po wyzerowaniu każdy μP z rodziny zachowuje się jak zwykły 8086. W architekturze 8086 można zauważyć odniesienia do 8080. Procesor dysponuje 20-bitową szyną adresową (wprowadzono multipleksowanie z danymi), a o tym, który bajt szyny jest aktualnie wykorzystywany decydują linie $A0$ oraz $\overline{BHE}$.
Ilość bajtów w rozkazie może wynosić od 1-8, nie występuje *wyrównanie*



Po resecie

IP=0000H
CS=FFFFH



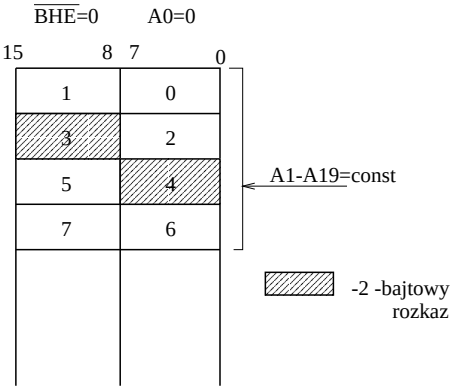
Rysunek 24: Rejestry 8086

kodu do słowa bądź długiego słowa, dlatego częstokroć następują pobrania słowa w dwóch cyklach magistrali – rys. 25.

Wersja 8088 różni się 8-bitową szyną danych i krótszą kolejką instrukcji (4-bajty).Ponieważ występuje kolejka instrukcji więc możemy zauważyć niejednoznaczność pomiędzy tym co dzieje się na magistrali, a tym co aktualnie robi μP . Mimo unieważnienia kolejki w przypadku rozkazów modyfikujących PC, sumaryczny efekt rozdzielenia części wykonawczej od części magistralowej jest pozytywny wskutek występowania wielu skomplikowanych instrukcji (podczas ich realizacji w klasycznym μP szyny pozostawały by niewykorzystane). Oczywistym jest, że dostęp do magistrali ma rów-

[3]Ryszard Goczyński, Michał Tuszyński *MIKROPROCESORY 80286, 80386 i i486*

nież układ wykonawczy jeśli zachopdzi konieczność wymiany argumentu z pamięcią bądź obszarem we/wy - priorytet układu wykonawczego jest wtedy większy. **μP 8086 może pracować w trybie minimalnym bądź maksymalnym**. Tylko w tym ostatnim jest możliwa praca wieloprocesorowa, a także proste podłączenie kooprocesora[4]. W trybie minimalnym μP samodzielnie wytwarza sygnały przeznaczone do sterowania magistrali oraz buforów zatrzaskujących adres i wzmacniaczy na szynie danych (odpowiednio $ALE, \overline{DLL}$ i $DT/\overline{R}$). W trybie maksymalnym sygnały sterujące magistralą $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}, \overline{INTA}$, a także sygnały sterujące buforami wytwarza współpracujący z μP sterownik magistrali 8288, jest on przystosowany także do współpracy z arbitrem w przypadku pracy wielu μP na wspólnej magistrali. Wszystkie sygnały sterownik 8288 wytwarza na podstawie zegara μP oraz podstawowych linii statusowych $\overline{S0} - \overline{S2}$ (patrz str.4 dodatku). Sterownik 8288 po stwierdzeniu przejścia statusu do jednego z aktywnych, rozpoznaje odpowiadający mu typ cyklu i generuje właściwą sekwencję sygnałów sterujących. Ewentualne wstawienie przez μP taktów oczekiwania objawia się dla sterownika wydłużeniem bieżącego statusu co powoduje automatyczne wydłużenie przezeń sygnałów sterujących.



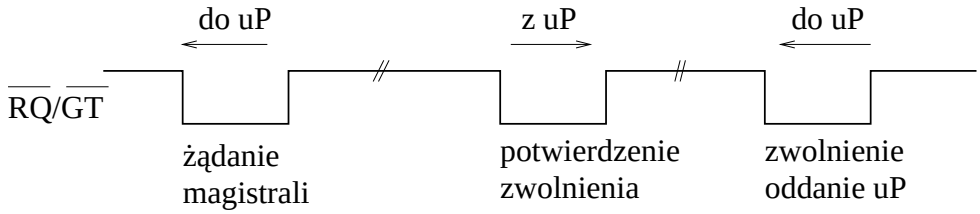
Rysunek 25: Pobranie rozkazu w 2-ch cyklach magistrali

Wyższe linie statusowe informują o używanym rejestrze segmentowym w danym cyklu, a S5 podaje sygnał zezwolenia na przerwanie równoznaczny bitowi **I** rejestru statusowego (bajt systemowy).
Odpowiednikiem podstawowych linii statusowych dla trybu minimalnego są linie $MEMR/\overline{I/O}, DT/\overline{R}, SS0$. Status ten występuje tylko dla wersji z 8-bitową szyną danych i został tak zmodyfikowany, aby można było wprost podłączyć jeszcze zintegrowane układy rodziny 8085.

O stanie kolejki informują linie $US0, US1$ **(tylko tryb max.)**. Linie te są niezbędne dla prawidłowej synchronizacji pracy zestawu z kooprocesorem. W trybie maksymalnym wprowadzono także niestandardowy protokół przejmowania magistrali (rys. 26. $\overline{RQ}/\overline{GT}$ dwukierunkowe.

Linia US0 ma wyższy priorytet. Linie kooprocesora mają zdolność tworzenia łańcuszka (tworzenia zapotrzebowania na magistralę). μP oddaje oczywiście magistralę po każdym cyklu, jedynie *nie mogą być rozerwane podwójne cykle akceptacji przerwania oraz cykle tych instrukcji, których KO został poprzedzony przedrostkiem LOCK.*

[4] Zbigniew Mroziński *Koprocesor arytmetyczny 8087*
Michał Tuszyński, Ryszard Goczyński *KOPROCESORY 80287, 80387 oraz i486*
Henryk Małysiak, Bolesław Pochpień, Eugeniusz Wróbel *Procesory arytmetyczne*



Rysunek 26: Przejmowanie magistrali

Niechęć oddania magistrali indykuje linia LOCK (tylko tryb max.) μP stosuje podwójny cykl akceptacji przerwania ze względu na dopasowanie się do ewentualnej konfiguracji ...sterowników przerwań 8259. Nr wektora jest zawsze dosyłany do μP w drugim cyklu, pierwszy cykl jest tylko wykorzystywany do zsynchronizowania sterownika *master* ze sterownikiem *slave*. Obsługa przerwań jest identyczna jak dla Z-80 z tym, że występuje tu 256 wektorów, a każdy zajmuje 4 bajty (segment i offset). Tablica wektorów nie może być przesuwana w przestrzeni adresowej lecz zajmuje lokacje od 0 – 3FF. W 8086 występuje tylko 5 wewnętrznych stanów wyjątkowych:

- nr 0 – dzielenie przez 0
- nr 1 – praca programowa krokowa
- nr 2 – NMI
- nr 3 – rozkaz INT3 (jednobajtowy)
- nr 4 – przekroczenie zakresu

Numery od 5 do 31 zostały zarezerwowane i są wykorzystywane sukcesywnie przez kolejne μP rodziny. Stan wyjątkowy dzielenia przez zero wykonuje się zawsze jeśli ten fakt nastąpi, podobnie jest dla NMI, natomiast stan wyjątkowy przekroczenia zakresu może być warunkowo zainicjowany jeśli μP odczyta rozkaz **INT0**. O tym czy stan wyjątkowy w programowej pracy krokowej będzie zgłaszany automatycznie po każdej instrukcji decyduje stan bitu T w bajcie systemowym słowa stanu. Podczas obsługi NMI fakt wystąpienia kolejnego aktywnego zbocza na tej nóżce jest pamiętany i nowa obsługa NMI może być rozpoczęta dopiero po wykonaniu IRET kończącego podprogram obsługi. Istnienie możliwościprogramowego zainicjowania dowolnej procedury stanu wyjątkowego używając rozkaz INT, z tym, że w odróżnieniu od przerwania zewnętrznego dostarczającego cyklu akceptacji numer rozkazu INT nie jest uzależniony od bitu *I* w bajcie systemowym, co więcej wykonanie rozkazu *INT nr*, kasuje ten bit. Jego stan będzie odtwarzany dopiero rozkazem *IRET* kończącym obsługę przerwania programowego. Po akceptacji przerwania wewnętrznego

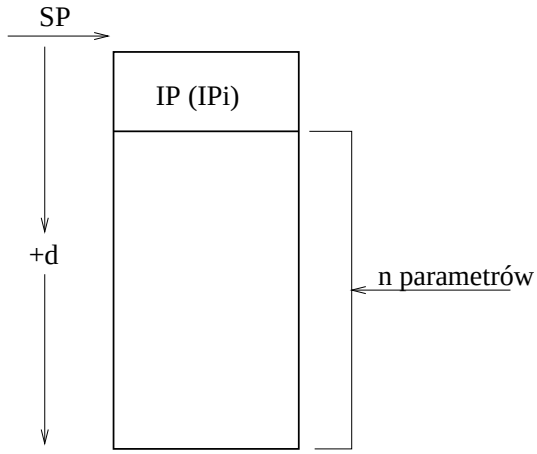
stanu wyjątkowego bądz po pobraniu rozkazu *INT nr* μP składa na stosie 6 bajtów:

- status (2 bajty)
- segment (2 bajty)
- przesunięcie (2 bajty)

Segment odpowiada zawartości rejestru *CS*, a przesunięcie to stan rejestru *IP*. Operacja odwrotna to instrukcja *IRET*. Wśród instrukcji *INT nr* wybrano rozkaz INT3 i zakodowano go na jednym bajcie (*INT3: CC*) przeznaczając nań oddzielny stan wyjątkowy. Instukcja ta jest przedewszystkim wykorzystywana do zakładania pułapek adresowych w wybranych miejscach na 1 bajcie KO. Ponieważ najkrótsze rozkazy 8086 są 1-bajtowe, więc taką samą długość musiała mieć instrukcja pułapki. Programowa praca krokowa zostaje zainicjowana po ustawieniu bitu T w bajcie systemowym.
Notabene ustawienie bitu T jest dość skomplikowane (zerowanie też :-)), można to zrobić jedynie drogą pośrednią np. przez następującą sekwencję instrukcji:

```
PUSH F
POP AX
OR AX, #0100
PUSH AX
POP F
```

Od tego momentu po zakończeniu każdego rozkazu μP będzie zgłaszał wewnętrzny stan wyjątkowy nr 1.
Należy zauważyć, że μP po zapisaniu na stosie aktualnego stanu bitów I oraz T kasuje ich stan wewnętrzny. Dzięki temu np. podczas obsługi rozkazu przerwania programowego nie są dopuszczane przerwania zewnętrzne, a podczas procedury programowej pracy krokowej jest wyłączane śledzenie.



Rysunek 27: Stos

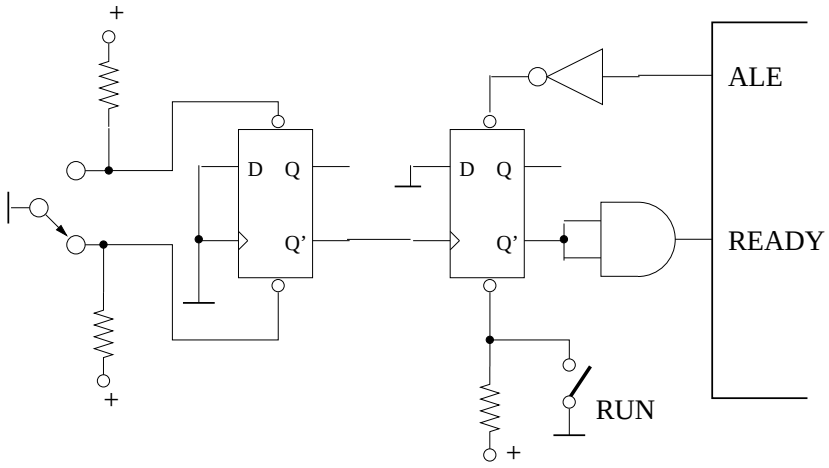
Segmentacja 8086 w konsekwencji prowadzi do występowania w przypadku skoków wywołań i powrotów do występowania wersji długiej i krótkiej. Rozkazy krótkie nie modyfikują rejestru CS, za wyjątkiem powrotu operują adresacją względną, a w przypadku wywołania i powrotu kładą i odczytują ze stanu jedynie dwubajtowe przesunięcie (rejestr IP). Wersje długie (pozasegmentowe) w przypadku skoku i wywołania muszą załadować nowe wartości do CS oraz IP, a wywołanie powrotu wykorzystują 4 bajty stosu (IP oraz CS). Można zauważyć równocześnie, że przy obu powrotach (bliskich i dalekich) występują także wersje z bezpośrednim zwiększeniem SP. Instrukcje te wykorzystuje się do usunięcia ze stosu parametrów pozostawionych przez podprogram, rys. 27. Naturalne użycie pewnych rejestrów segmentowych może być modyfikowane przedrostkiem segmentu, którym można wymusić użycie innego rejestru segmentowego (rys. 9 pobrań operacji stosowanych ich przy adresie przeznaczenia operacji łańcuchowych zmiany segmentu są niedopuszczalne (pozostają odpowiednio CS, SS i ES).

Rodzaj operacji	segment	segment	offset
	podstawowy	alternatywny	
pobranie	C	–	IP
stosowe	S	–	SP
inne	D	C,E,S	EA
łańcuchy źródło	D	C,E,S	SI
łańcuchy przeznaczenie	E	–	DI
BP jako rejestr bazowy	S	D,C,E	EA
BX jako rejestr bazowy	D	C,E,S	EA

Tabela 9: Możliwe kombinacje rejestrów

4.2 Karta CPU

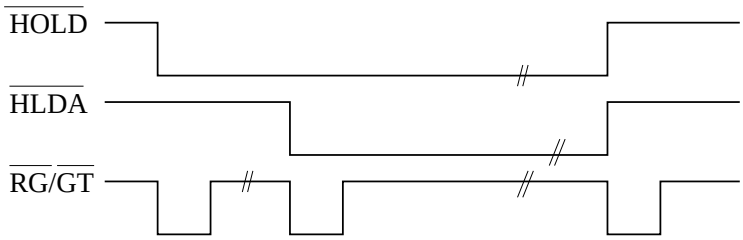
μP pracuje równolegle z kooprocesorem, zestaw ten do komunikacji z systemem poprzez magistralę wymaga wzmacniaczy pamiętających na adresie



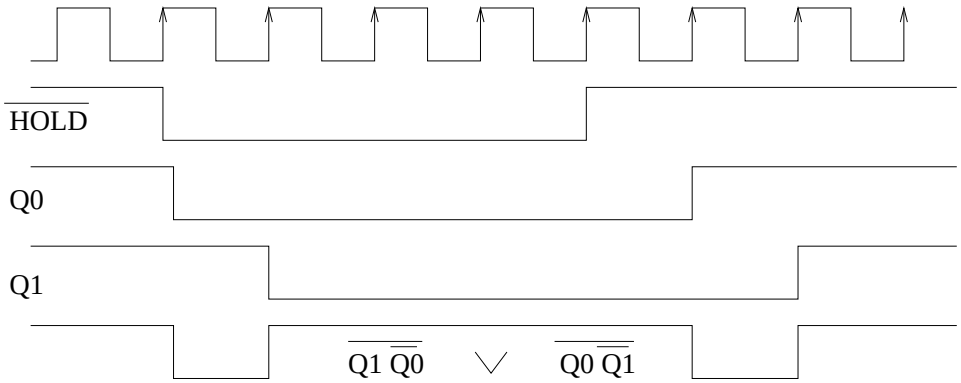
Rysunek 28: Praca krokowa

i wzmacniaczy dwukierunkowych na danych. Adres jest ważny w momencie opadającego zbocza ALE. W trybie maksymalnym sygnały sterujące zatzaskami i wzmacniaczami wytwarza sterownik 8288 (ALE, DT/R, DEN). Dla zegara dla μP , kooprocesora i systemu jest wytwarzany w dedykowanym układzie 8284, w którym również synchronizuje i wspomaga logikę niegotowaści i zerowania.
Na karcie z logiką linii READY skojarzono również układ do pracy krokowej (rys. 28).

Kooprocesor 8087 rozpoznaje czy współpracuje '86 czy '87 po stanie linii $\overline{BHE}/S7$ (8088 wyprowadza tam status pomocniczy $\overline{SS0}$). Kooprocesor na bieżąco śledzi działanie μP głównego dzięki równoległemu połączeniu oprócz adresu i danych także linii statusowych S0-S2 i kolejki QS0-QS1. Jeśli w ciągu rozkazów pojawi się instrukcja ESC to μP na podstawie jej drugiego bajtu (jeśli to jest potrzebne) wylicza adres argumentu i dokonuje odczytu jego początkowego fragmentu. 8087 śledząc na bieżąco pracę μP również rozpoznaje rozkaz ESC. Kooprocesor *przechwytuje* adres początku argumentu poczym logikę linii RQ/GT zawiesza μP . Po zawieszeniu 8087 doczytuje pozostałą część argumentu bądź zapisuje cały argument. Na czas wykonywania operacji arytmetycznej 8087 aktywuje linie BUSY. Linię tę łączymy z wejściem TEST μP . Jeśli wynik obliczeń zmiennoprzecinkowych jest zaraz potrzebny μP -owi to po rozkazie ESC umieszczamy instrukcję WAIT (próbkujące wejście TEST). Po odczytaniu argumentu '87 oczywiście zwolnił magistralę. W przypadku stwierdzenia nieprawidłowości funkcjonalnych lub numerycznych '87 może zgłosić przerwanie. Jego linie statusu



Rysunek 29:

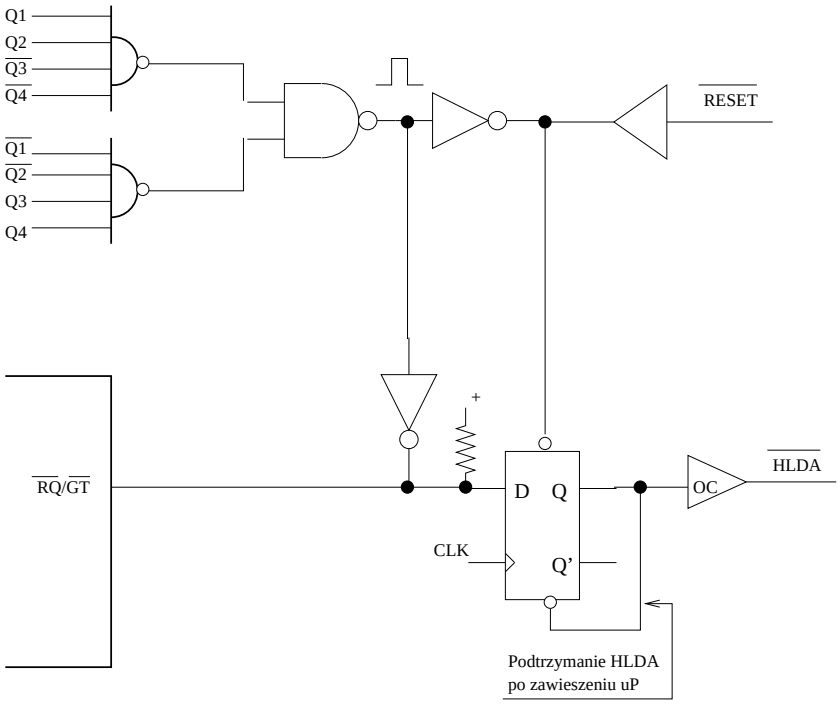


Rysunek 30: Generacja impulsu

jeśli są ustawione na zewnątrz, a nie na nasłuch mogą oczywiście generować wyłącznie statusy zapisu i ODCZYTU PAMIĘCI.
Niewykorzystaną linią przez kooprocesor RQ/GT0 w '87 jest przeznaczona dla żądań zawieszenia μP od innych układów (np. sterownika DMA). Zaszła konieczność zamiany tej dwukierunkowej logiki na jednokierunkowe HOLD i HLDA (rys. 29).

Jak widać każda zmiana stanu HOLD musi wygenerować impuls na linii RQ/GT.

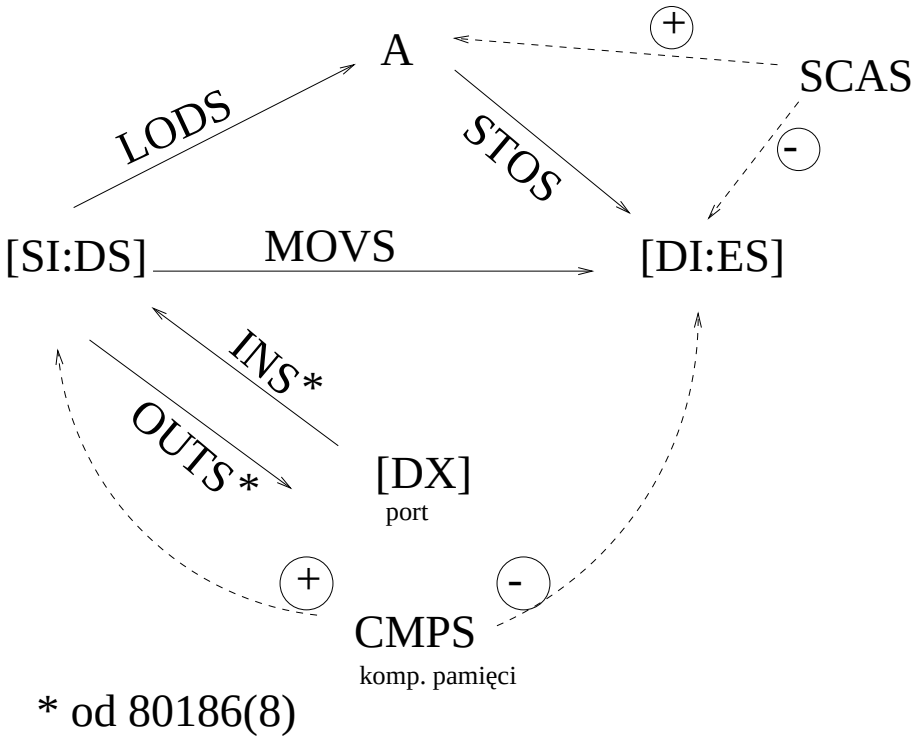
Prosta generacja impulsu o długości jednego okresu zegara przy pomocy dwóch przerzutników (rys. 30 spełnia postawione wymogi należy jednak zauważyć, że pojawienie się zakłócenia na aktywnym *zerze* akurat w trakcie aktywnego zbocza zegara wywoła generację błędnego impulsu. Stosując opóźnienie przy pomocy 4 przerzutników zabezpieczamy się przed tym efektem jeśli tylko impuls nie będzie dłuższy od okresu zegara.


Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
RESET
RQ/GT
+
D
Q
CLK
Q'
OC
HLDA
Podtrzymanie HLDA
po zawieszeniu uP

Rysunek 31:

4.3 Specyfika listy rozkazów

W związku z segmentacją adres efektywny wynikający z kodu operacyjnego (16-bitowy) musi być odniesiony do rejestru segmentowego (przeważnie jest to rejestr DS). Sposób wyznaczenia EA (adres efektywny) określają pola MOD, REG, R/M przy ewentualnym wykorzystaniu bitu W(tab. str8). Bez przedrostka długość rozkazu może wynieść 6-bajtów. Analizując pola kodów operacyjnych możemy zauważyć za jednym argumentem musi być rejestr (nie jest możliwa operacja na dwóch lokacjach pamięciowych). Oprócz rozkazów ustawiających pojedyncze bity statusowe (CARRY, DIRECTORY, INTERUP) są możliwe przesłania wyłącznie bitów warunkowych z akumulatorem. Przestrzeń we/wy jest 64-bajtowa. Jedynie adresacja przez DX umożliwia pełne jej wykorzystanie, rozkazy IN i OUT dwubajtowe na drugim bajcie zawierają wyłącznie 8-bitowy adres. W przypadku, gdy w rozkazach IN i OUT jest ustawiony KO bit W, przesłanie słowa odbywa się zawsze po D0-D7 w dwóch cyklach(w drugim cyklu adres jest incrementowany).
W obrębie segmentu w przypadku skoku i wywołań używane są przeważnie rozkazy z adresacją względną (dla skoku także z przesunięciem 8-bitowym). Jeśli sięgamy poza segment to rozkazy te muszą za KO podawać 4-bajty nowego offsetu (IP) i segmentu(CS). Ponadto możliwa jest adresacja pośrednia z wykorzystaniem pól *mody* oraz *rm*, dla skoków krótkich określa się w ten sposób początek dwóch, a dla długich początek czterech kolejnych bajtów w pamięci w których zapisany jest offset bądź offset i segment.
W związku z krótkimi i długimi wywołaniami są też wyłącznie bezwarunkowe długie i krótkie powroty odtwarzające za stosu dwa bajty IP bądź 4 bajty IP i CS. W przypadku powrotów długich i krótkich występują też mutacje dodające do Sp wartość bezpośrednią drugiego i trzeciego bajtu. Można je wykorzystywać w celu usunięcia ze stosu pozostawionych tam przez podprogram parametrów. Skoki warunkowe (takie jak w 6800) obejmują wszystkie relacje pomiędzy liczbami ze znakiem i bez znaku, a także od stanu poszczególnych bitów. Mają one wyłącznie 8-bitową adresację względną. Do tworzenia pętli programowej można wykorzystać instrukcje LOOP i jej mutacje, są to przypadki poznanej uprzednio instrukcji DB.
Ręczne zamykanie pętli umożliwia instrukcja JCXZ, rejestr CX jest tutaj wyłącznym rejestrem zliczającym podobnie jak przy instrukcji LOOP. Oprócz przerwań zewnętrznych możemy je emulować programowo instrukcjami INT (nie występuje cykl akceptacji, a nr wektora nie jest dosyłany z zewnątrz, lecz precyzuje go drugi bajt rozkazu).
Przerwanie programowe związane z 3-wektorem dysponuje krótkim jed-



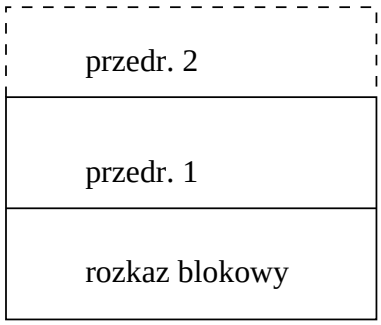
Rysunek 32: Przesłania

nobajtowym kodowaniem, instrukcja ta została wprowadzona dla realizacji programowych pułapek systemowych. Rozkaz INT0 uruchamia procedurę związaną z 4-wektorem w przypadku jeśli bit przekroczenia będzie ustawiony (identycznie jak TRAPV). Każda procedura obsługi każdego przerwania kończy się rozkazem IRET odtwarzającym ze stosu 6-bajtów (flagi, segment i offset).
W 8086 występują przesłania i porównania blokowe z tym, że nie są one tak jak w Z80 poprzedzielane "ślepymi" pobraniami KO. Aby pewne instrukcje mogły być powtarzane muszą zostać poprzedzone przedrostkiem powtórzenia (bezwarunkowym lub warunkowym).

REP #F3 (MOVS, STOS, INS, OUTS)
(LOADS - bez sensu z REP)

(dla SCAS, CMPS)
REPE - aż do CX=0 lub Z=0 (#F2)
REPNE - aż do CX=0 lub Z=1 (#F3)

Akceptacja przerwań w przypadku instrukcji blokowych może nastąpić po każdym "nawrocie". Np dla MOVS po każdej parze odczyt-zapis, a dla SCAS po każdym odczycie. Używając instrukcji blokowych musimy naprzód ustalić adres źródła, przeznaczenia, wpisać liczbę nawrotów do CX, a także sprecyzować przy pomocy bitu D kierunek zmian adresu argumentu.



Rysunek 33: Przedrostki

Oddanie magistrali przy operacjach blokowych następuje wg. powszechnie obowiązujących zasad. Przerywając operację blokową musimy oczywiście zachować na stosie wszystkie parametry stringu w celu późniejszego jego kontynuowania. Po wejściu w przerwanie μP kładzie na stosie adres pierwszego przedrostka instrukcji blokowej, w którym zwykle jest REP. Powoduje to, że jeżeli przedrostkiem drugim (rys. 33) będzie LOCK to po powrocie z przerwania μP nie będzie chronił operacji blokowej lecz na żądanie zwróci magistralę. Zamiana przedrostków tj. REP, LOCK instrukcji blokowej nic nie daje, a wręcz blokuje string po powrocie z przerwania. Adres efektywny może być też wyłącznie wyznaczony i zapamiętywany w rejestrze w celu poźniejszego wykorzystania. Rozkaz ten LEA oraz dwa następne LDS, LES są pożyteczne przy tworzeniu procedur relokowalnych.

LEA:
(reg) $\leftarrow$ EA
LDS:
reg $\leftarrow$ (EA)
DS $\leftarrow$ (EA+2)
LES:
reg $\leftarrow$ (EA)
ES $\leftarrow$ (EA+2)

W przypadku przekodowywania wartości nie przekraczających 1-bajtu możemy zainstalować w pamięci 256-bajtową tablicę, której początek wskazuje BX, a indeksem jest dola połowa akumulatora.

$$(BX + AL) \leftarrow AL$$

W μP 8086 występują cztery typy przerwań:

- programowe (INT nr), bądź przerwanie dla dzielenia przez 0
- NMI
- INT
- śledzenie (przerwanie od pracy krokowej)

Zostały one uszeregowane wg. malejących priorytetów. Przerwanie INTR jest jest maskowane bitem I, a przerwanie od śledzenia bitem T. Dwa pierwsze rodzaje nie są od niczego uzależnione. Wszystkie przerwania powodują natomiast skasowanie bitu I oraz T. W środku μP -ra na stosie zostaje złożona ich poprzednia wartość przywracana rozkazem IRET. Przerwanie programowe składa się z następujących cykli:

1. pobranie
2. pobranie, gdy INT3
3. 4 $\times$ odczyt wektora
4. 4 $\times$ zapis stosu

porozdzielanych ewentualnie cyklami magistrali wynikającymi z działania kolejki. W przypadku przerwania zewnętrznego występuje ta sama liczba cykli.

- akceptacja (1-sze INTA) [aktywny LOCK]
- akceptacja (2-gie INTA) [aktywny LOCK], tu aktywny jest nr wektora
- 4$\times$ odczyt wektora
- 6$\times$ zapis stosu

W przypadku zainicjowania śledzenia będzie ono wykonywane po każdej instrukcji nawet jeśli w jej trakcie μP musi zaakceptować INT przy aktywnym I bądź stwierdził aktywne zbocze NMI.

4.4 Tryb wirtualny 386

Rozwiązanie to jest kolejnym po 286 μP INTEL'a. Procesor ten posiada 32-bitową architekturę, szynę adresową i danych, a także możliwość pracy wieloprocesorowej z dostępem do tzw. przestrzeni wirtualnej. Podobnie jak 80286 i następne posiada 4 poziomy uprzywilejowania.

0 - najwyższy, wewnętrzny

1 -

2 -

3 - użytkownika

Pełny dostęp do wszystkich zasobów jest możliwy oczywiście na poziomie 0. μP 386 pozwala na bezpośrednie przeniesienie zadań napisanych dla 80286 (rozróżnia formaty stosu i segmentu zadań), a także może bezpośrednio w trybie wielozadaniowym wykonywać programy napisane dla 8086. Należy zaznaczyć, że wszystkie instrukcje swoich poprzedników zawarte są w liście rozkazów tego μP .
80386 nie dysponuje wewnętrzną pamięcią podręczną (cache) lecz dla jej wykorzystania musi posługiwać się odpowoednim sterownikiem. Pamięć wewnętrzną wprowadzono dopiero do μP 80486, w swoim wnętrzu zawiera on także koprocesor arytmetyczny identyczny z 80387.
W związku z 32-bitową architekturą i magistralami zamiast linii A0 i A1 pojawiły się linie zezwoleń ($\overline{BE}$) dla poszczególnych bajtów szyny danych. Linie $W/\overline{R}$, $D/\overline{C}$ i $M/\overline{IO}$ ($zapis - \overline{odczyt}$), $dana - \overline{kod}$, $pami - \overline{we - wy}$) pozwalają na zdekodowanie 8 typów cykli magistrali (analog do S0-S2 w 8086).
Wyjaśnienia wymagają pewne linie sterujące:

- $\overline{ADS}$ - strob adresu, indykuje rozpoczęcie każdego cyklu magistrali
- $\overline{IN}$ - wymusza zakładkowość ??????? na pamięci
- BS16 - przełącza szynę danych na transfery 16b (w 486 jest też BS8)
- BUSOFF - zabiea magistralę po każdym cyklu zegara (takcie)

Znaczenie $\overline{READY}$, HOLD, HLDA, INTR, NMI oraz $\overline{LOCK}$ identyczna jak w poprzednich procesorach. Należy tylko zauważyć, że zakres zastosowań przedrostka LOCK jest ograniczony wyłącznie do operacji typu *odczyt-modyfikacja-zapis*. Niewłaściwe użycie generuje wyjątek niezidentyfikownej instrukcji.
Zarówno dane,a tym bardziej kod nie mają wyrównania binarnego. Możliwość tę wprowadzono dopiero w 486 i to wyłącznie z oczywistych przyczyn (zgodność listy rozkazów) tylko dla danych. Cykl magistrali tego μP składa

się z 2-ch taktów T_1 (wystawiony ADS) i T_2. W przypadku konieczności wydłużenia występuje zwiększanie ilości T_2. Próbkowanie linii $\overline{READY}$ dokonywane jest na końcu każdego T_2 i w zależności od stanu tej linii ustawiany jest kolejny T_2 bądź zaczyna się nowy cykl T_1.
Odczytywane dane mogą być ważne dopiero na końcu cyklu (????? $T_2 i T_1$ następnego). Opóźniony odczyt wprowadzono ze względu na możliwość stosowania pamięci o dłuższych czasach dostępu. Temu samemu celowi służy też praca *"zakładkowa"* wymuszona aktywnym stanem $\overline{NA}$. W tym trybie sygnały adresu, typu cyklu i $\overline{BE_n}$ razem ze strobem $\overline{ADS}$ są wyprzedzone o 1 stan, tak że wymagany czas dostępu do pamięci wynosi zamiast 2 trzy (3) takty. Akceptacja przerwania (podwójny cykl INTA) jest realizowany identycznie jak w 8086, z tym, że na podstawie A_2 możemy rozpoznać pierwsze i drugie INTA.
Procesor 286 i dalsze po włączeniu zasilania bądź po sygnale RESET rozpoczynają pracę w trybie 8086 (szybkość pracy jest oczywiście większa). W związku z tym dostęp jest wyłącznie do przestrzeni 2^20 bajtów.
Kontrolę nad całą przestrzenią adresową μP uzyskuje po przejściu w tryb tzw. wirtualny (wielozadaniowy). Program w trybie rzeczywiatym (przed przełączeniem) wykonuje się na 0-wym poziomie uprzywilejowania. Przejście d trybu wirtualnego następuje po wpisaniu "1" do bitu PE w rejestrze CR0. Powrót ewentualnie w tryb rzeczywisty realizujemy tą samą drogą (80286 tylko na drodze zerowania).
Po włączeniu zasilania '386 wystawia adres FFFFFFF0, rozpoczynając z pod tego adresu pobranie 1-szej instrukcji, po następującym zawsze długim skoku następuje ustalenie zgodnie ze skokiem linii $A_2 - A_{19}$, natomiast wyższe linie przechodzą w stan 0. (procesor pracuje na 0-wej 1MB stronie adresowej).

4.4.1 Adresacja w trybie wirtualnym

8192^{13} – deskryptorów 8-mio bajtowych

Selektor – wskaźnik do tablicy deskryptorów

W trybie rzeczywistym adres fizyczny był budowany na podstawie 16-bitowego segmentu i 16-bitowego przesunięcia. Adres segmentu zawierał się w rejestrze segmentowym, a przesunięcie było w:

- IP
- SP
- wyliczane na podstawie stanu rejestrów wg. użytego trybu adresacji.

W trybie wirtualnym filozofia adresacji jest inna. Używa się pojęcia przesunięcia i selektora. Obie te wartości precyzują lokacje pamięci. Przesunięcie ma znaczenie jak wyżej, natomiast selektor informuje względem jakiego segmentu należy użyć przesunięcia. Każde zadanie dysponuje własnymi segmentami:

- kodu (1)
- danych (1 - 4)
- stosu (1)

Miejsce położenia w pamięci operacyjnej tych segmentów, ich rozmiar oraz atrybuty zawarte są w tzw. deskryptorze segmentu. Deskryptory ulokowane są w tablicach deskryptorów:

- globalnej (wspólnej),
- lokalnych, opcjonalnych dla każdego zadania

Wartość zapisana do rejestru segmentowego w trybie wirtualnym nie jest więc adresem segmentu, lecz wskaźnikiem do deskryuptora opisującego dany segment. Adres powstały na drodze odniesienia przesunięcia do adresu początku odpowiedniego segmentu nie musi być jeszcze adresem fizycznym tak jak jest to w 286. W ogólnym przypadku może to być tzw. adres liniowy (2^{32}). Dopiero podanie adresu liniowego mechanizmowi stronicowania wygeneruje docelowy adres fizyczny. Strony 80386(486) mają stałą wielkość 4kB. Użycie mechanizmu stronicowania ułatwia wymianę segmentów pomiędzy pamięcią dyskową, a pamięcią operacyjną adresowaną bezpośrednio przez μP . Przede wszystkim zaś stronicowanie zapobiega tzw. (de)fragmentacji zbiorów dyskowych, a także pozwala optymalnie wykorzystać pamięć operacyjną. Jest to tak jakby konteneryzacja. Wpisana nowa wartość do rejestru segmentowego jest w trybie wirtualnym traktowana przez μP jako tzw. selektor, tj jako wskaźnik do odpowiedniej tablicy deskryptora. (Format deskryptora- str.6)
Selektor wskazuje na opis segmentu czyli tzw. deskryptor segmentu w tablicy deskryptorów. W deskryptorze zawarta jest informacja czy dany segment (danych, stosu, kodu) jest obecny (odwzorowany) w przestrzeni pamięci operacyjnej, a jeśli tak to od jakiego adresu się znajduje i jaką wielkość przestrzeni zajmuje. Jeśli w danym momencie segment jest tylko na dysku, a nie w pamięci operacyjnej, to opisujący go deskryptor ma wstawiony bit jeden atrybutu P=0. Sięgnięcie do takiego segmentu przez μP spowoduje wygenerowanie stanu wyjątkowego braku segmentu.
Obsługująca go procedura ma za zadanie zorganizować wolne miejsce w pamięci operacyjnej i w to miejsce ściągnąć żądany segment. Po jego ściągnięciu procedura musi uaktualnić deskryptor, tj. wpisać P=1 i zaktualizować

jego położenie, czyli tzw. bazę względem której będzie liczone przesunięcie. Aby wygospodarować wolne miejsce w pamięci operacyjnej procedura systemowa przeprowadza co jakiś czas analizę wykorzystania segmentu. Na tej podstawie można usunąć z pamięci operacyjnej segment najdawniej bądź najrzadziej używany. Do analizy tej używany jest bit A w polu atrybutu segmentu. μP sięgając po dany segment może go wyłącznie ustawić (segment użyty). System operacyjny co jakiś czas przegląda bity A w deskryptorach (opisach segmentów) z ich równoczesnym kasowaniem i na tej podstawie może podjąć decyzję o usunięciu segmentu z pamięci operacyjnej. W segmencie usuniętym zostaje wyzerowany bit P. Oznacza to, że jest on nieobecny, a wartość bazowa nie ma znaczenia (przeważnie jest błedna). Stosując stronicowanie musimy dysponować tablicami informującymi o tym, gdzie dana strona 2^{32} adresu liniowego jest ulokowana w pamięci fizycznej (2^{32}). μP 80386 dysponuje mechanizmem umożliwiającym szybkie przełączanie zadań, tj. przydzielenie CPU innemu zadaniu, a zawieszenie zadania poprzedniego. W klasycznym μP do tego cel był wykorzystywany stos. Tutaj natomiast następuje to bez użycia instrukcji zapisu-odczytu niejako automatycznie, a więc szybciej.
Każde zadanie może dysponować segmentem stosu zadania, tj. (segment systemowy) w którym μP zapisuje cały swój stan w momencie zawieszenia danego zadania. Swój nowy stan pobiera z segmentu nowego zadania. Należy zauważyć, że wielkości adresów w segmencie stanu zadania (TS) są zapisane w postaci przesunięcie-selektor. Przykładowo dla kodu mamy określone wartości wznowienia w postaci *EIP-CS*. Po przełączeniu zadań (pobraniu nowego stanu z TS) μP musi więc odczytać deskryptor z segmentu kodu (wg. selektora z CS), dopiero dodanie EIP do adresu początku tego segmentu utworzy adres instrukcji, od której rozpocznie się dalsza obsługa wznowionego zadania (pod warunkiem wyłączonego stronicowania). Podobniejak tablica lokalna i globalna deskryptorów w trybie wirtualnym wyjątki korespondują z 256-pozycyjną tablicą deskryptorów przerwań. Jej adres początkowy i rozmiar zapisane są w rejestrze IDTR. Deskryptor w tablicy IDTR są wyłącznie deskryptorami systemowymi.
W czasie przełączania zadań .....deskryptory TSS są oczywiście zaznaczone jako zajęte. Mechanizm ten jest szczególnie przydatny w systemie wieloprocesorowym kiedy dane zadanie może być obsługiwane w różnych momentach czasu przez różne wolne procesory. Aktualne wykonywane zadanie przez dany μP jest oznaczone jako zajęte, jego obsługa nie może być przechwycona przez dany μP . Ponadto mechanizm ten jest niezbędny w przypadku wywoływania jednego zadania przez drugie. W tej sytuacji zarówno zadanie wywołujące jak i wywoływane muszą być zaznaczone jako zajęte. Zapobiegamy w ten sposób sytuacji, że zadanie wywołane odwołuje się

z powrotem do wywołującego, np: W trakcie wykonywania zadania w rejestrze TPR znajduje się selektor (nr) związany z deskryptorem systemowym TSS. Deskryptor ten informuje gdzie zaczyna się i jaką ma wielkość tzw. segment stanu zadania. Oczywistym jest, że kopia tego deskryptora znajduje się w μP w rejestrach niedostępnych programowo skojarzonych z rejestrem TR. W trybie wirtualnym wszystkie rozkazy wykonują się pominąwszy rozmiar argumentu identycznie jak w rzeczywistym za wyjątkiem:

- dalekich skoków [międzysegmentowe]
- dalekich wywołań [międzysegmentowe]
- dalekich powrotów [międzysegmentowe]
- instrukcji *INT nr* oraz *INT 3*
- instrukcji *INT0*
- instrukcji *BAUD*
- instrukcji *IRET*

Jeśli teraz μP pobierając rozkaz tego skoku sięgnie do tablicy deskryptorów GDT i stwierdzi, że w tym miejscu znajduje się deskryptor TSS, to w konsekwencji zainicjowany zostanie proces przełączania zadań. W tym przypadku 4-bajtowe przesunięcie zostanie zignorowane. Wartość EIP μP odczyta sobie bowiem z segmentu stanu zadania.
Po odczytaniu deskryptora stosu TSS nowe nowego zadania zostaje on zaznaczony jako zajęty (od tego momentu zajęty jest segment zadania zawieszonego i wznowionego). μP występuje teraz do zachowania swojego stanu w TSS zadania zawieszonego. W tym celu przepisuje stan swoich rejestrów (od EIP do LDT). Jest to stan, jaki był tuż przed wykonaniem instrukcji JMP. Następnie z TSS zadania wznowionego odczytuje wartości startowe rejestrów od CR3 do T. Dodatkowy odczyt CR3 jest związany z ewentualnym używaniem stronicowania, w CR3 zawarty jest adres katalogu translacji danego zadania. Odczyt bajtu z TSS z offsetem 64 informuje μP , czy ustawiony jest bit T pułapki przełączania zadania (.....stanu wyjątkowego nr 1 zmiany zadania). Po odtworzeniu stanu wznowionego zadania μP może zaznaczyć w deskryptorze zadania zawieszonego, że jest ono wolne. Należy zaznaczyć, że w rozkazie JMP pole przesunięcia zostaje zignorowane. Faktyczna wartość EIP μP odczytuje z segmentu stanu zadania. Oprócz przesunięcia EIP są tam zawarte, także przesunięcia względem początku stanu ESP oraz EBP, a także źródła przesunięć w stosunku do bazu segmentu danych (pozostałe rejestry ustawione są w sposób zależny od trybu adresacji). Dlatego też μP musi jeszcze określić bazy od, których będą te przesunięcia odnoszone.

Bazy są oczywiście zawarte w opisach odpowiednich segmentów, dlatego μP musi załadować szereg deskryptorów. Ponieważ każdy ich selektor zawiera 1 bitowe pole określające, czy odnosi się on do tablicy globalnej, czy lokalnej, dlatego też pierwszym odczytanym deskryptorem będzie odczytywany z tablicy globalnej deskryptor LDT (w rejestrach .....μP zapamięta bazę i!rozmiar tej tablicy). Ponieważ na etapie przełączaia zadań w każdym momencie μP może stwierdzić różne nieprawidłowości (wyjątki) dlatedo też ...po LDT ładowany jest deskryptor stosu. Jest to stos poziomu użytkownika (3-go) na którym wykonywane jest każde zadanie użytkownika. Należy zauważyć, że segment TSS zawiera też informacje dla μP o parametrach stosów wyższych poziomów (w formie selektor-przesunięcie). Dopiero teraz μP może odczytać deskryptor kodu. Zanim jeszcze zacznie go wykonywać ładuje aktywne selektory danych i oznacza je jako zajęte. (*Jeśli dany segment danych nie jest używany zapisujemy ten selektor 0-wy*). Dopiero teraz μP może po zsumowaniu bazy, segmentu kodu i przesunięcia EIP pobrać pierwszą instrukcję wznowionego zadania. Odczytując deskryptor stosu, kodu i danych oznacza je jako użyte ustawiając w nich A=1. (typy deskryptorów pamięci str.6)
Jeśli bit $\overline{S} = 1$ oznacza to, że dany deskryptor jest deskryptorem pamięci (kody, stosu, danych). 3-b pole typu precyzuje możliwości wykorzystania takiego deskryptora.
W szczególności istnieje możliwość .....segmentu danych przed zapisem. Oprócz tego możemy zabezpieczyć segment kodu przed odczytaniem (skopiowaniem, μP dopuszcza wtedy tylko jego wykonanie). Ze względu na to, że stos narasta w kierunku malejących adresów wprowadzono segmenty rozszerzalne w dół. W przypadku przekazywania parametrów przez stos ...zablokować możliwość ich zmiany (segment rozszerzony w dół - tylko odczyt). W danej chwili μP znajduje się na tzw. bieżącym poziomie uprzywilejowania (CPL). Jak wiadomo poziomu CPL możemy przenieść się przy wywołaniu na poziom .... Jest to związane z obniżeniem CPL (0-wy poziom ...uprzywilejowany). Jeśli jednak wywołany segment kodu będzie segment zgodny, to nie nastąpi ewentualna zmiana CPL. Warunkiem poprawności jest oczywiście, aby pozom segmentu zgodnego był wyższy od poziomu CPL (segment ...2-go poziomu można ...z poziomu 3 i 2-go, segment zgodny 0 z każdego). Jeśli chcemy przykładowo ...kod, to selektor jego segmentu musimy umieścić w jednym z rejestrów segmentu danych.
Rozróżniamy następujące rodzaje poziomów uprzywilejowania:

- CPL – bieżący
- RPL – żądający, w selektorze
- DPL – poziom adresu segmentu - w deskryptorze

- IOPL – uprzywilejowanie operacji we/wy

W szczególności w przypadku dostępu do segmentu danych wymaga się, aby zostały spełnione następujące nierówności:

1. $DPL \geqslant CPL$
2. $DPL \geqslant RPL$

Pierwsza nierówność jest oczywista, natomiast drugą wprowadzono wyłącznie na możliwość blokowania dostępu do danych przez zadania słabo uprzywilejowane, ale mające dostać się do ważnych danych dzięki obniżeniu CPL po wywołaniu procedury systemowej. W przypadku dostępu do kodu musi obowiązywać: $CPL_2 \leqslant CPL_1$. Zależność ta nie musi być spełniona w przypadku ...zadań w skutek ich pełnej separacji. Nie obowiązuje to jeśli zachodzi przypadek wywołania zadania przez zadanie.
μP 8086 w 256 elementowej tablicy były zdefiniowane adresy początków programów obsługi dla 5 wewnętrznych stanów wyjątkowych, a także (od 32-255) pozycje przeznaczone dla obsługi przerwań (lokacje do 31 zostały zarezerwowane dla przyszłych wyjątków wewnętrznych). W trybie rzeczywistym '386 startuje z położeniem i rozmiarem tej tablicy identycznym jak dla 8086 (baza=0, ...=3FF). W trybie wirtualnym pod każdym numerem tablicy nie może znajdować się 46 pole segment-offset, lecz 8B deskryptor dlatego, że przed przejściemdo trybu wirtualnego przeważnie ustala się nowe położenie tablicy IDT zapisując jej rejestr IDTR oraz zapisuje się W wybranym przez rejestr obszarze pamięci deskryptory używanych procedur i spodziewanych wyjątków wewnętrznych. W tablicy IDT mogą znajdować się jedynie 3 typy deskryptorów tzw. systemowych:

1. pułapki (stanu wyjątkowego)
2. przerwania
3. zadania

Deskryptory te posiadają inny format w stosunku do deskryptorów pamięci. Format tych deskryptorów systemowych jest różny w stosunku do deskryptorów pamięci. W szczególności deskryptor z tablicy IDT zawiera pełne przesunięcie (miejsce wejścia do programu obsługi) niezbędne .....) Selektor ten wskazuje na docelowy deskryptor z programu obsługi stanu wyjątkowego bądź przerwania. Adres pierwszego rozkazu zostanie wyznaczony przez sumowanie przesunięcia deskryptora systemowego z bazą odczytaną z deskryptora segmentu kodu. W szczególności w systemie może znajdować się segment kodu wspólny dla obsługi wszystkich stanów wyjątkowych. Natomiast w różnych deskryptorach systemowych będą zaznaczone odpowiednie różne przesunięcia w tej procedurze (segment) w takiej sytuacji

wszystkie deskryptory systemowe odnoszące się do wspólnego segmentu z obsługą wyjątków będą miały wpisaną tę samą wartość w polu selektora. Oprócz odczytania deskryptora systemowego i deskryptora kodu przed rozpoczęciem obsługi wyjątku μP musi jeszcze zapisać na stosie *"namiary"* powrotu. Ponieważ jest to też stan wyjątkowy bądź przerwanie, dlatego też zapisywane są oprócz wielkości segmentu i przesunięcia także wartości rejestru statusowego. Podobnie jak dla '86 obsługa stanu wyjątkowego czy przerwania musi zakończyć się instrukcją IRET. Na razie założono milcząco, że nie zachodzi zmiana poziomu CPL i nie jest konieczne przełączanie stosów. Obsługa stanu wyjątkowego, czy przerwania może być też zrealizowana niejako przez oddzielne zadanie. W tym przypadku czytając deskryptor systemowy (w skutek wystąpienia wyjątku bądź przerwania) stwierdzi w polu atrybutów typ 5 (brama zadania). W takiej sytuacji istotną część deskryptora będzie wyłącznie, oprócz atrybutów, selektor. Musi on odnosić się do deskryptora segmentu DSS i musi być to segment aktualnie dostępny. Obsługując wyjątek bądź przerwania przez oddzielne zadania pomijamy pole przesunięciadeskryptora bramy zadania, ponieważ wartość przesunięcia odczytamy wprost z TSS.
Do tej pory zakładano, że obsługując wyjątek nie zmieniamy poziomu CPL, to znaczy rejestr statusowy i lokację powrotną μP zapisał na stosie zadania, które wyjątek spowodowało (poziom 3).Sytuacja taka jednak występuje sporadycznie, gdyż zwykle obsługa wszystkich wyjątków obsługi jest na poziomie 0-wym. W związku z tym zachodzi konieczność tzw. przełączenia stosów. Każdy poziom musi posługiwać się swoim własnym stosem, W związku z tym przy operacjach stosowych wymaga się pełnej zgodności wszystkich poziomów uprzywilejowania: $CPL = RPL = DPL$.
Jeśli zachodzi zmiana poziomu uprzywilejowania, to należy również przejść na stos bardziej uprzywilejowany (właściwy dla obsługi wyjątku). Parametry stosów poziomów 0-2 są zapisane w TSS danego zadania (zadanie wykonuje się na poziomie 3, a parametry stosu poziomu 3 są w bieżących rejestrach SS i ESP μP). Po odczytaniu parametrów nowego wewnętrznego stosu μP ładuje jego deskryptor (typ danych rozszerzalny w dół, odczyt i zapis). Na nowym stosie zapisywane są teraz najpierw wartości rejestrów ESP i SS z μP właściwe dla poziomu 3-go. Dopiero teraz, na nowym stosie można zapisać rejestr statusowy, CS i EIP właściwe dla instrukcji, która owodowała wyjątek. Instrukcja IRET odczytując standardowo lokację powrotną i ...stwierdza, czy zachodzi konieczność zwrotnego przełączania stosu. Jeśli tak, to μP odczytuje kolejne 2 długie słowa po czym sięga po deskryptor poprzedniego stosu celem załadowania z niego nowej wartości ESP. Wszystkie

operacje na stosie odbywają się przy aktywnym wyjściu $\overline{LOCK}$ (μP przez ten czas nie potwierdza żądania HOLD). Oprócz tego linia LOCK jest aktywna przy operacjach na tablicach, translacji (stronicowanie) oraz identycznie jak w 8086 ...cyklami INTA i w przypadku użaycia przedrostka LOCK zakres jego ...jest tu ograniczony wyłącznie do sytuacji uzasadnionych, tzn. rozkazów modyfikujących argument w pamięci (nap. NEG). Nieodpowoednie użycie przedrostka LOCK spowoduje zgłoszenie wyjątku 6-go nielegalnej instrukcji, na stosie μP położy adres przedrostka LOCK. W odpowiedniej pozycji tablicy IDT skojarzonych z numerem dosyłanym z IDT w 2 cyklu INTA możemy umieszczać deskryptory systemowe bram przerwania, pułapki (stanu wyjątkowego) lub zadania. Stosując bramę przerwania powodujemy wygenerowanie bitu IF, natomiast wejście do procedury obsługi poprzez bramę stanu wyjątkowego nie kasuje IF. Stan bitu IF przy wejściu poprzez bramę zadania zależeć będzie natomiast od wartości zapisanej w TSS i ładowanej przy wejściu w procedurę obsługi. Należy zauważyć, że posługiwanie się bramą wyjątku bądź przerwania gwarantuje szybką reakcję na przerwanie (także za ...). Natomiast obsługa przez oddzielne zadanie jest dłuższa, z tym, że możemy zapewnić całkowitą separację i nie musimy niczego zachowywać na stosie. Rozkazy odnoszące się do bitu IF podobnie jak instrukcje IN, INS, OUT, OUTS są zależne od poziomu ochrony we/wy (pole IOPL w rejestrze statusowym). Wykonanie tych instrukcji jest dopuszczalne jeżeli tylko poziom bieżący $CPL \leqslant IOPL$ (jest bardziej ważny od IOPL). Przykładowo ustawiając IOPL=0 dopuszczamy modyfikację bitu IF tylko na poziomie 0 (zadania nie mogą go zmienić). Dokładniej istnieje możliwość elastycznego przydziału poziomu IOPL poprzez odpowiednie ustawienie tego pola w poszczególnych TSS'ach. W celu dalszego uelastycznienia w dostępie poszczególnych zadań do zasobów systemowych we/wy definiuje się w każdym TSS bitową mapę zezwoleń (1 bit dla każdego z 64k lokacji i przestrzeni we/wy). Jeśli bit danej lokacji = 0 , to operacje we/wy wykonują się bezwarunkowo. Jeśli bit jest ustawiony to przed wykonaniem rozkazu we/wy sprawdzana jest nierówność: $CPL \leqslant IOPL$. Oprócz omawianych bitów IOPL oraz znanych z μP 8086 w rejestrze statusowym '386 pojawiły się nowe. Istnieją następujące możliwości zmiany bitów statusowych:

- 0-7 – rozkaz przesłania $\leftrightarrows$ AL, poza tym wykonanie operacji przez ALU
- 0-15 – poprzez stos, dotyczy to głównie bitów 8-15
- 0-31 – instrukcja IRET, ponadto wszystkie bity statusowe są modyfikowane w procesie przełączania zadań

FLAGS - bity

NT – wskaźnik zadania zazgnieżdżonego - 1: gdy jedno zadanie wywo-

łuje inne; przejście do zadania połączone z zagnieżdżeniem dokonuje instrukcja międzysegmentowa CALL, natomiast powrót z zadania zagnieżdżonego powoduje rozkaz IRET. Ponieważ IRET kończy też obsługę stanu wyjątkowego, dlatego o sposobie wykonania go decyduje bit NT. Wykonanie IRET kasuje bit NT.

RF – bit ten zapobiega wielokrotnej generacji stanu wyjątkowego związanego z pułapką ustawioną w jednym z rejestrów DR. Po stwierdzeniu wyjątku dla danej instrukcji bir RF=1. Jego skasowanie nastąpi dopiero wtedy, kiedy instrukcja wykona się prawidłowo, tzn zostaną usunięte wyjątki z nią związane.

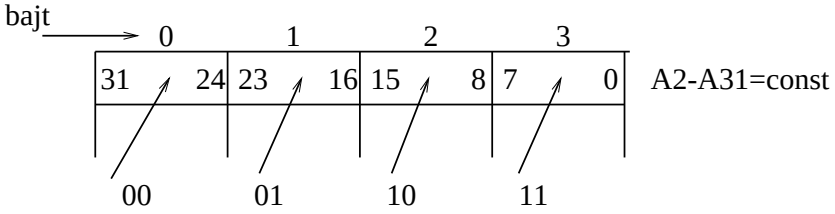
VM – bit trybu wirtualnego 8086. Ustawienie tego bitu umożliwia wykonywanie zadań 8086 w trybie wirtualnym. Zadania te korzystają z 1MB segmentów i zapewniona jest pełna przenoszalność kodu

AC – znacznik sprawdzenia wyrównania. Pwoduje zgłoszenie wyjątku przy dostępie niewyrównanym do danych (np. jeśli adres 32 argumentu nie jest podzielny przez 4). Zgłoszenie wyjątku nr 17 jest możliwe tylko na poziomie 3 przy równoczesnym ustawieniu bitu maskującego Am w rejestrze sterującym CR0.

5 MAGISTRALA VME

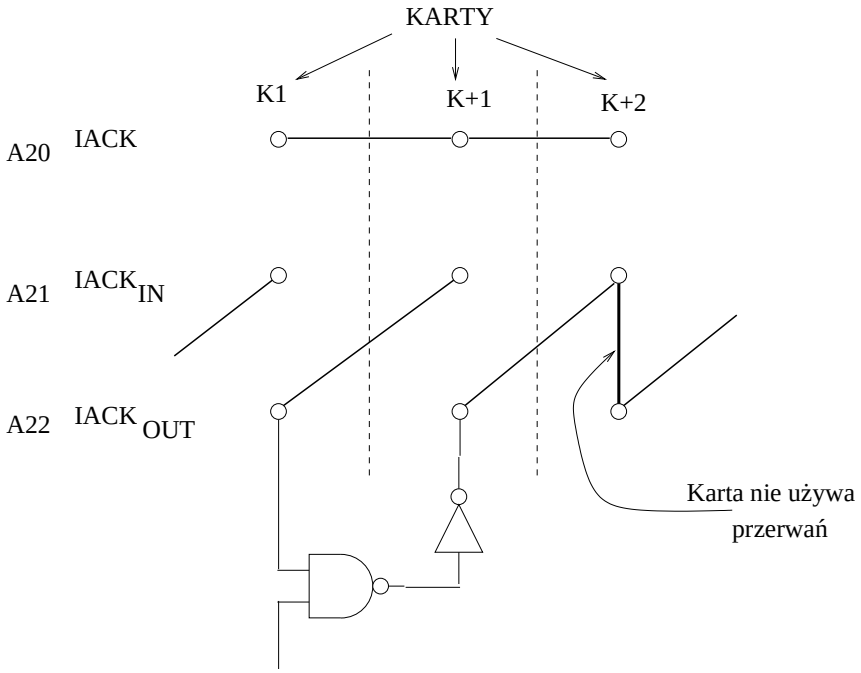
5.1 Opis funkcjonalny magistrali VME

Magistrala ta została opracowana głównie z myślą o μP Motoroli, lecz jest wykorzystywana dla innych CPU. Szczególnie jest wykorzystywana w układach automatyki przemysłowej. Oprócz maksymalnie 32-bitowego adresu zdefiniowane są modyfikatory adresu **AM0 - AM5**, które są tak jakby rozszerzeniem adresowym (podobnie jak przestrzenie definiowane liniami FC dla 68000). Linie AM definiują także jaka część z liniiA_1-A_{31} jest aktualnie wykorzystywana. Kolejność poszczególnych bajtów w obrębie 32-bitowej szyny danych wygląda następująco: O tym jaka część szyny danych (1,2, 3



Rysunek 34: Szyna danych

lub 4 bajty) jest wykorzystywana informują linie **A1**, $\overline{LWORD}$ oraz $\overline{DS0}i$ $\overline{DS1}$. Przy każdym transferze w dowolnym formacie chociaż jedna z tych ostatnich linii jest aktywna, ponieważ są to linie strobu danych.
Należy zwrócić uwagę, że strob ważności adresu **AS** potwierdza również ustalony stan na liniach AM0 - AM5, $\overline{LWORD}$ i lini $\overline{WRITE}$. Dopiero po przejściu na 0, $\overline{AS}$ może wejść w stan aktywny - linia (linie DS). Sygnały magistrali są zgrupowane na dwóch 96 - pinowych złączach, górnym J1 i dolnym J2. Można zauważyć, że w przypadku przesłań pojedynczych bajtów oraz słów (za wyjątkiem słowa 1 - 2) wykorzystuje się zawsze linie D0 albo linie D15 (ze złącza J1). Jeśli tylko użyte CPU nie używa więcej lini adresowych jak 24 pozwala to wtedy na zupełną eliminację dolnego złącza J2. Złącze to zawiera górną połowę szyny danych, najstarszy bajt adresowy i szereg linii definiowanych dla własnych potrzeb dla konkretnego użytkownika. Odnośnie przerwań standart dopuszcza 7 - poziomów zgłoszeń **IRQ1-7**. W przypadku akceptacji przerwania przez CPU wytwarza ono sygnał potwierdzenia. W cyklu z aktywnym sygnałem **IACK** jest na liniach D0-D7 odbierany od urządzenia przerywającego bajt z numerem wektora. O tym czy dane urządzenie, które zgłosiło przerwanie IRQ_i wyśle numer wektora decyduje jeszcze poziom akceptacji ze strony CPU (wypro-



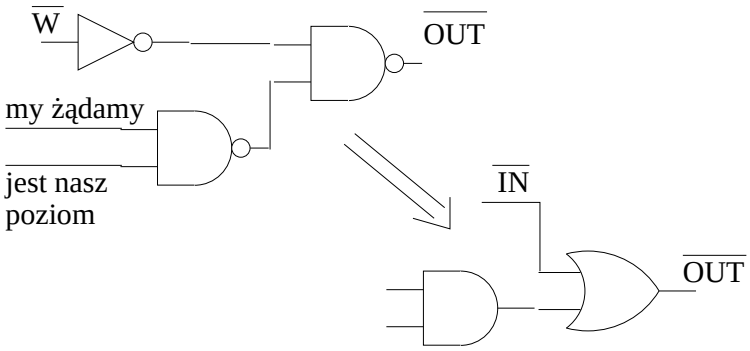
Rysunek 35: Linia łańcuszkowa

wadzany na A1 - A3), a w obrębie tego samego poziomu położenie karty w kasecie (im bliżej CPU tym priorytet w obrębie tego samego poziomu będzie wyższy). Jest to technicznie rozwiązane za pomocą łańcuszkowej linii (rys. 35) $\overline{IACK_{IN}} - \overline{IACK_{OUT}}$.

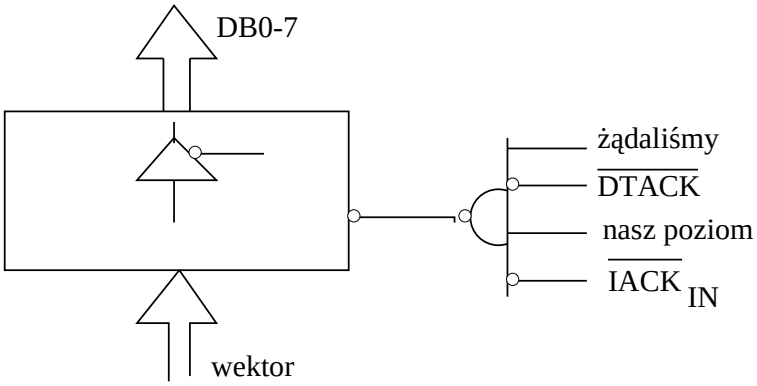
Odnośnie lini łańcuszkowej obowiązują następujące reguły gry:

- jeśli dana karta zgłosiła żądanie obsługi to może wystawić swój wektor tylko wtedy jeśli stwierdzi swój poziom, na którym żądała (A1 - A3 w cyklu z aktywnym IACK) oraz dotrze na jej $IACK_{IN}$ stan aktywnego zera.
- jeśli karta nie wystawiała żądania, albo nie pojawił się jej poziom to stan na linii $IACK_{IN}$ jest transmitowany na $IACK_{OUT}$. Karta nie używana zwiera te piny, a w miejscu gdzie karta nie jest zainstalowana należy je zewrzeć na magistrali. Po odebraniu 1 na IN, każda karta musi ją przetransmitować na OUT (rys. 36).

- KI = 1 - arbiter przyznał magistralę kanałowi na poziomie **i** (i=0 do 3). Każda karta, która może przejąć funkcje zarządzania magistralą (master), przy zapotrzebowaniu magistrali wysyła poprzez bramkę OC aktywny stan na odpowiedniej linii BR (na danym poziomie może żą-



Rysunek 36:



Rysunek 37: Przyznanie magistrali

dać więcej niż 1 - karta). Zgłoszenie BR odbiera arbiter umieszczony w początkowym slocie (zerowym) magistrali.

- Przyznając magistralę jednej z kart wysyła on na łańcuszkową linię potwierdzenia **BG0 - 3** aktywny niski stan. Arbiter może stosować priorytety sztywne (najważniejszy poziom trzeci) lub priorytet blokujący (karta na danym poziomie po wykorzystaniu magistrali przechodzi na poziom najniższy).
- Karta która korzysta z magistrali utrzymuje aktywny niski stan na linii **BBSY**. Jeśli podczas swojej pracy stwierdzi, że na linii **BCLR** pojawi się zero oznacza to, że arbiter żąda od niej zwolnienia magistrali. Po zakończeniu aktualnego cyklu karta taka dezaktywuje swoje bufory 3-stanowe, którymi wysterowała magistralę i zwalnia linię BBSY (powraca ona w stan wysoki). Arbiter po stwierdzeniu nieaktywnego stanu na $\overline{BBSY}$ może przydzielać magistralę na innym poziomie posyłając aktywne zero do odpowiedniej linii łańcuszkowej $\overline{BG}$.
- Jeśli na danym poziomie żądają magistrali dwie lub więcej karty to potwierdzenie BG przechwyci karta ulokowana najbliżej arbitra. Dlatego przedstawiono sieć działania arbitra magistrali. Arbiter taki odbiera żądanie BR oraz odczytuje stan linii BBSY. Sygnałami wyjściowymi arbitra są linie $\overline{BG}$ oraz $\overline{BCLR}$. O tym na którym poziomie została przyznana magistrala decyduje jedynka wpisana do odpowiedniego przerzutnika K0 - K3 na karcie arbitra. Sieć działania dotyczy arbitra ze stałym priorytetem.
- W przypadku priorytetu rotującego będzie obowiązywał inny wewnętrzny algorytm, a ...nie będzie używał linii $\overline{BCLR}$. Karty które nie korzystają z dostępu do magistrali transmitują tylko stan BG, podobnie zachowują się karty w stosunku do poziomu przezeń nieużywanych. Na indywidualnej karcie, która korzysta z magistrali jako master musi znaleźć się prosta logika transformująca sygnały BR i BG (IN i OUT) oraz BBSY i BCLR np. na sygnały danego μP (HLDA, HOLD, BR, BG, BGACK itp.).

6 ORGANIZACJA SYSTEMÓW

6.1 Przykłady realizacji kart funkcjonalnych systemów

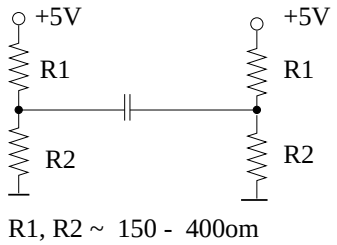
Karty pamięci - układy pamięciowe scalone oprócz odpowiedniej liczby linii adresowych (jednokierunkowych) oraz linii danych (przeważnie dwu-

kierunkowych, wersje z rozdzielonymi liniami we-wy spotyka się rzadko) posiadają sygnały sterujące:
Linie adresowe mogą być multipleksowane co występuje w przypadku pamięci DRAM, zupełnie inne jest też wtedy sterowanie takich układów (sygnały $\overline{BAS}, \overline{CAS}, \overline{WE}$). Tworząc kartę pamięciową oprócz odpowiedniej ilości układów pamięciowych musimy na niej zawrzeć jeszcze następujące podzespoły:

- dekoder adresowy
- układy sterujące
- bufory szyny danych
- logika dopasowująca czas dostępu do czasu cyklu magistrali

Dekoder adresowy stwierdza czy aktualny adres na magistrali dotyczy układów zainstalowanych na danej karcie. W przypadku dekoderów przy pamięciach w zależnie od ich pojemności na wejście dekodera nie jest podawana pewna pewna ilość młodszych linii adresowych. Oprócz linii adresowych dekoder wykorzystuje również linie potwierdzające ważność adresu ($\overline{AS}, VMA$, itp.) W układach dekodera adresów mogą również wchodzić przełączniki relokujące dany moduł w obrębie przestrzeni adresowej systemu.
Układy sterujące dopasowują standard sygnałów sterujących magistrali do funkcji podanych powyżej wejść pamięci ($\overline{MEMR}, \overline{MEMW}, \overline{E}, R/\overline{W}, DS0, DS1, \overline{MREQ}$). Bufory pośredniczące pomiędzy wyjściem układów pamięci, a szyną danych są w większości przypadków niezbędne ze względu na ograniczoną obciążalność układów pamięciowych. W cyklu odczytu układy te musiały by wysterować wprost szynę danych do, której przecież podłączone jest wiele różnych kart (wymogi prądowe i czasowe wynikające z pojemności pasożytniczych).

Odpowiednia obciążalność prądowa w cyklu odczytu każdej karty wynika przede wszystkim z istnienia tzw. terminatorów (rys. 38) na końcu magistrali. Układy te zapobiegają odbiciom, a także przesłuchom pomiędzy liniami. Oczywistym jest, że ich zastosowanie wiąże się ze zwiększeniem wydajności prądowej buforów szczególnie w niskim stanie logicznym. Częstokroć magistralę łączącą poszczególne karty wykonuje się drukiem wielowarstwowym, w którym linie sygnałowe przedzielone są liniami masy. Ilość bitów bufora szyny danych przeważnie równa się ilości bitów szyny



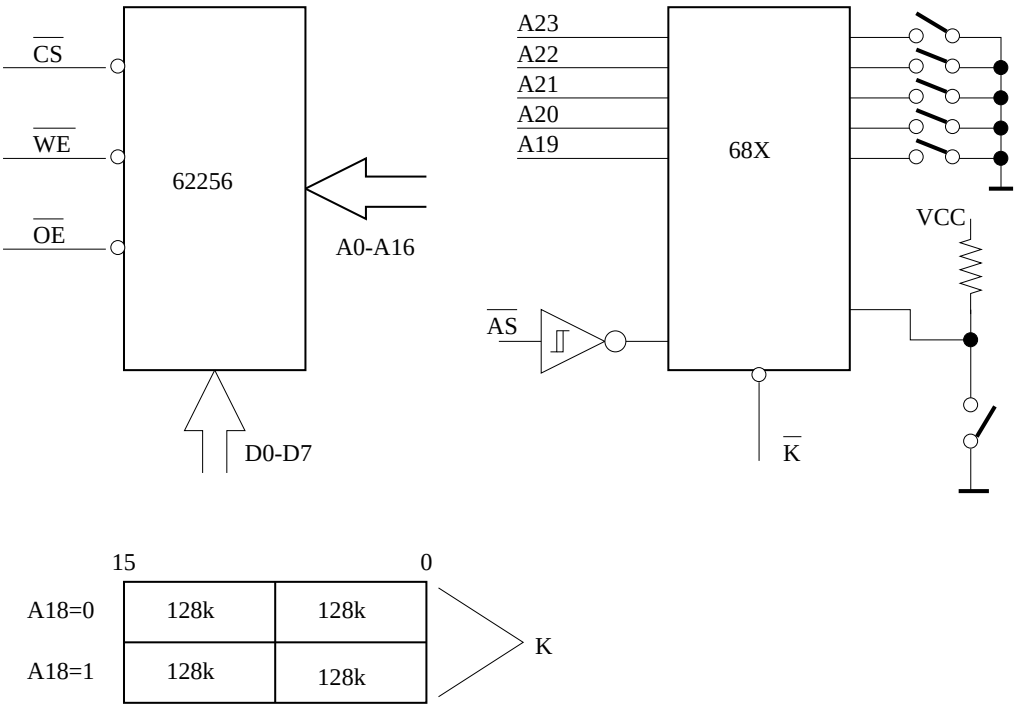
Rysunek 38: Terminatory

danych magistrali. Od reguły tej zdarzają się wyjątki. Szczególnie w kartach we/wy spotykane są bajtowe dostępy do magistrali, pozostała część słowa w cyklu dostępu bywa niewykorzystywana.
Niektóre rozwiązania wyposażone w logikę dynamicznej zmiany szerokości szyny danych pozwalają na stosowanie kart o pełnej szerokości szyny danych, lecz o odpowiednio *węższej* bramie dostępu.
Np. program napisany dla μP 32-bitowego zamiast rozłożenia go na cztery kości bajtowe może być odczytywany z pojedynczej pamięci w porcjach 8-bitowych – μP każdy cykl 32-bitowy *rozmnoży* na cztery podcykle, a kolejno odczytane bajty skieruje do właściwej lokacji np. linijka programu.
Przykładem takiego rozwiązania są linie μP 80486 i 68030. Można zauważyć, że dla magistrali **VME**, jeśli przewidujemy różne warianty konfiguracji odczytywanych i zapisywanych bajtów, niezbędnym będzie użycie 6-dwukierunkowych buforów 8-biowych. Dwa dodatkowe muszą być zastosowane ze względu na to, że przesłania o szerokości bajtu bądź słowa wykorzystują zawsze linie D0-D15. W większości wspólczesnych rozwiązań czas reakcji pamięci jest decydujący o szybkości pracy systemu. Jeśli użyte układy są wystarczająco szybkie w systemie normalnie gotowym nie jest wymagana logika oczekiwania. Logika taka nazywana tutaj logiką potwierdzenia jest natomiast wymagana zawsze niezależnie od szybkości pamięci w systemach normalnie niegotowych.
Powyższe układy po rozpoznaniu adresu karty pamięci wymuszają odpowiednio niski stan na linii niegotowości (typu OC), bądź po odpowiednim czasie *ściągają na zero* linie potwierdzającą.

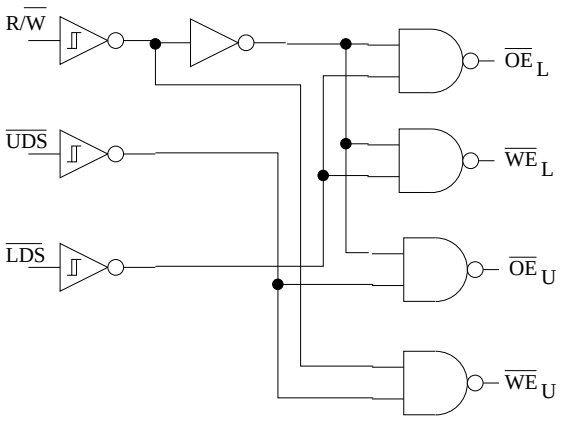
6.2 Karta pamięci dla systemu 68000 (8086)

$\overline{CS}$	$\overline{CE}$	$\overline{WE}$	Opis
1	X	X	nieaktywny
0	1	1	aktywny, brak operacji
0	0	1	odczyt
0	1	0	zapis
0	0	0	stan zabroniony

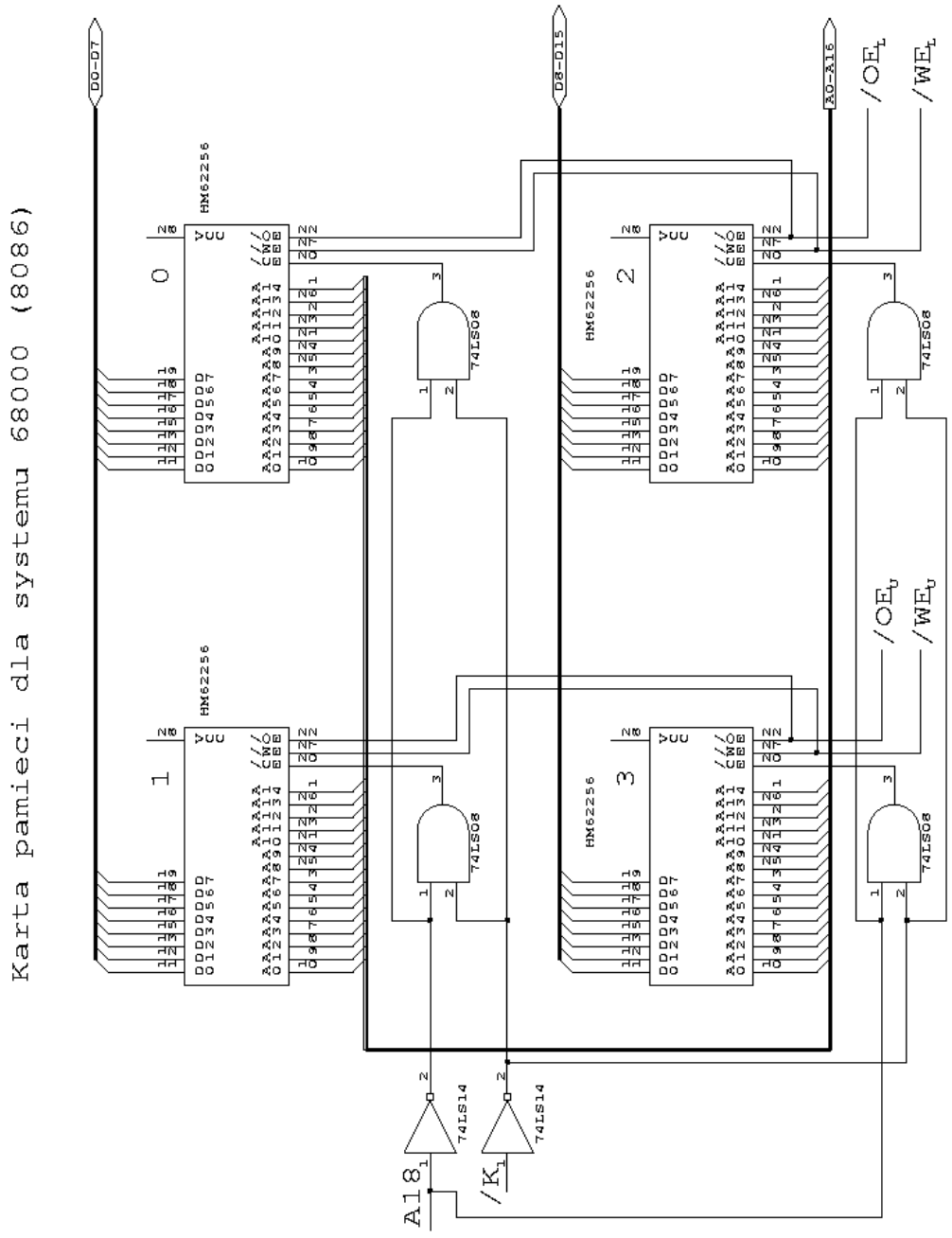
Tabela 10:



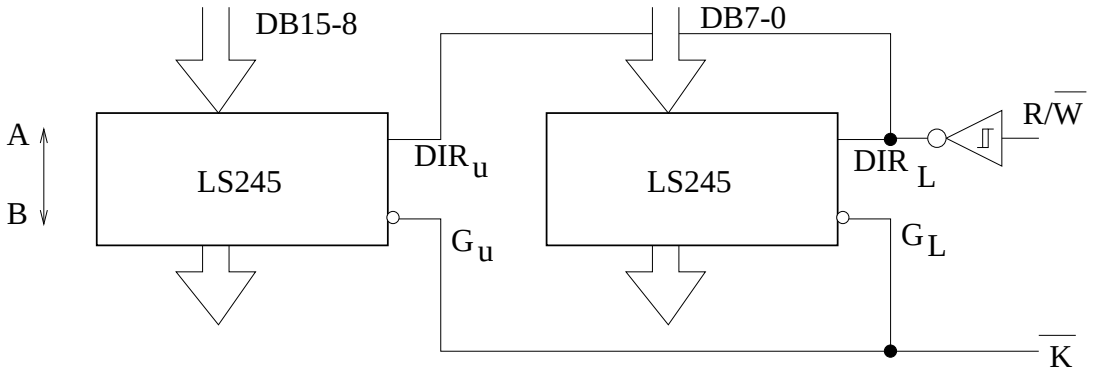
Rysunek 39: Zarys karty



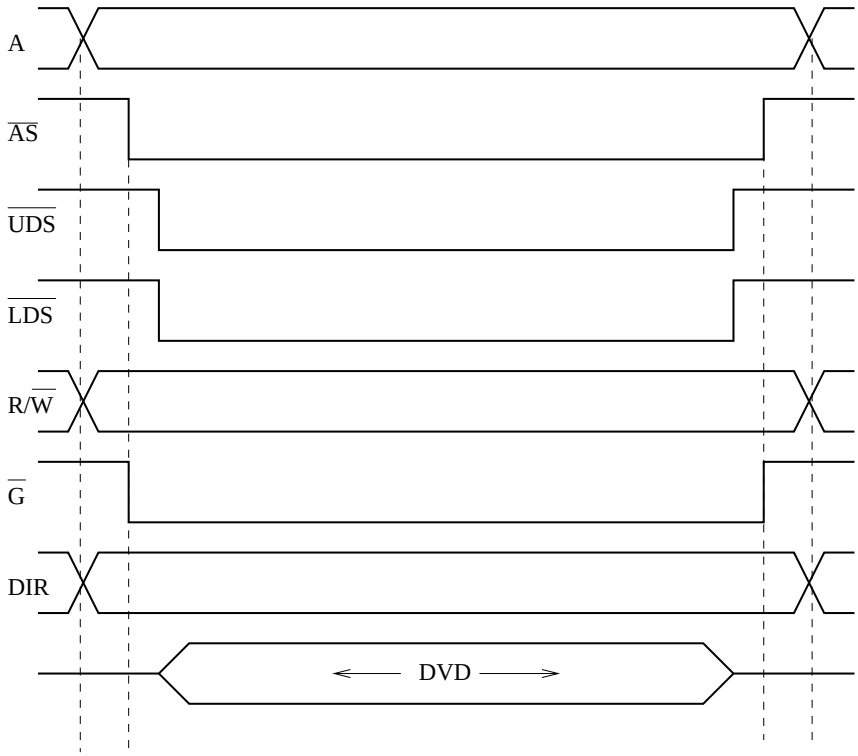
Rysunek 40: Generacja sygnałów sterujących



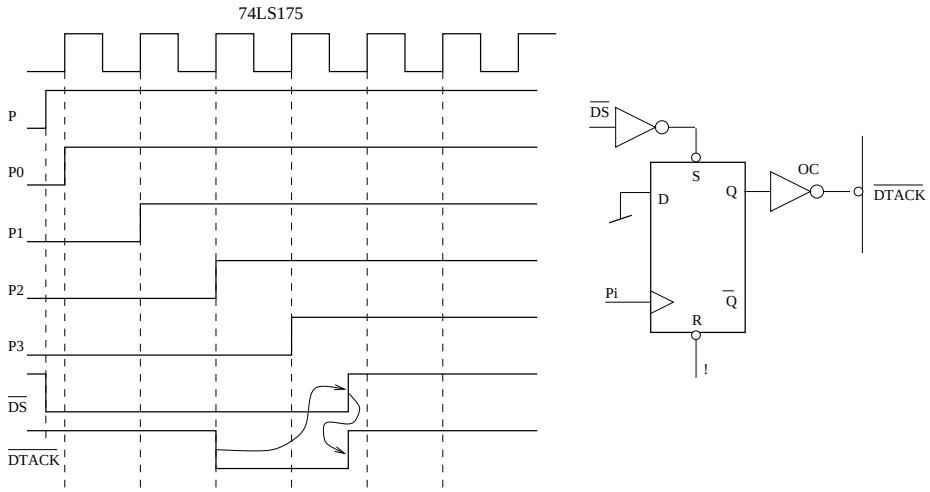
Rysunek 41: Schemat ideowy karty



Rysunek 42: Bufory pośredniczące



Rysunek 43: Przebegi czasowe



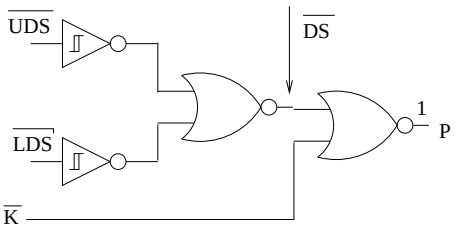
Rysunek 45: Logika potwierdzenia

$\overline{G}$	DIR	
1	X	Z
0	1	A $\rightarrow B$
0	0	B $\leftarrow A$

Tabela 11: Sterowanie przepływem LS245

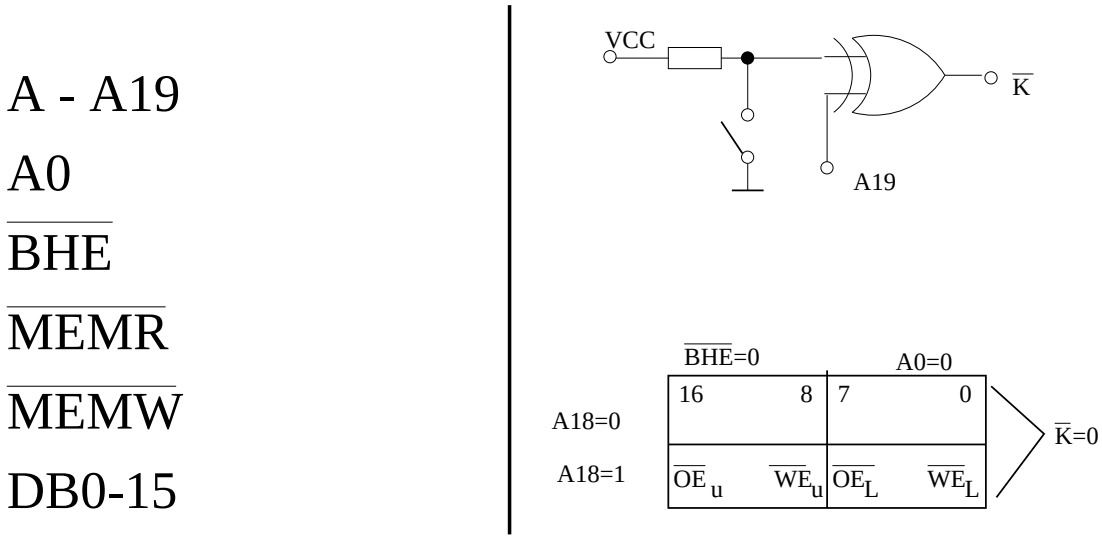
Sterując bufory nie rozróżniamy przy ich aktywacji linii UDS i LDS. Powoduje to, że bufory są aktywne przez cały czas $\overline{AS} = 0$ tzn. *szerzej w stosunku do UDS i LDS*. Jeśli μP chce odczytać pojedynczy bajt to wskutek otwarcia obu LS245 otrzyma wprawdzie na drugim bajcie same jedynki, ale nie będą one nigdzie wykorzystywane. W przypadku natomiast zapisu do pamięci, dane pojawią się też na wejściach obu kostek niewykorzystywanego przez μP bajtu, ale odnośna linia $\overline{WE}$ kostek 1,3 bądź 0,2 nie będzie wtedy aktywna i zapis nie nastąpi.

Inicjacja układu potwierdzenia sygnałem $\overline{DS}$ (rys. 45, a nie AS wynika z faktu istnienia cykli *odczyt-modyfikacja-zapis* wymagający podwójnego potwierdzenia w trakcie tego samego AS. W przypadku konieczności współpracy powyższej karty z 8086 należy dokonać pewnych modyfikacji wynikających z przyjętej tam konwencji sygnałów.



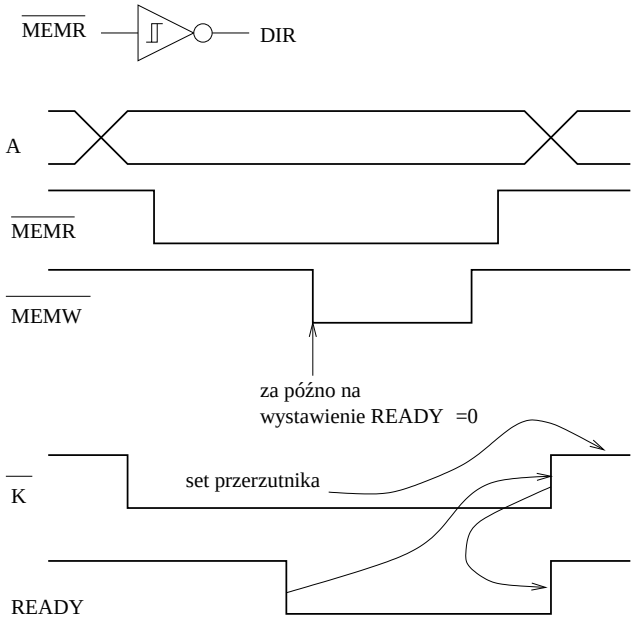
Rysunek 44:

Cykle *odczyt-modyfikacja-zapis* na 8086 nie są konieczne, gdyż μP ten dysponuje przedrostkiem LOCK. Po umieszczeniu go przed rozkazem μP w czasie jego trwania nie reaguje na żądanie oddania magistrali (linia HOLD w trybie minimalnym lub $\overline{RG}/GT$ w trybie maksymalnym). W związ-

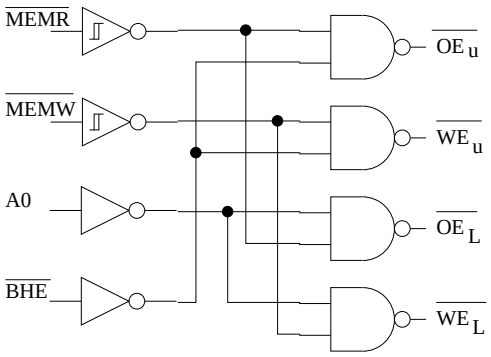


Rysunek 46: Współpraca z 8086

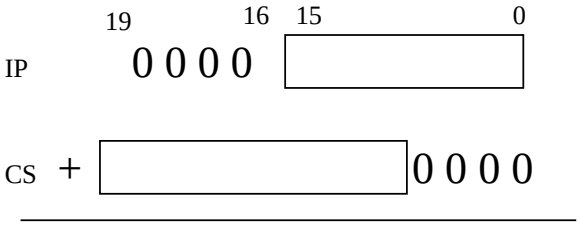
$$\overline{G_u} = \overline{G_L} = K$$



Rysunek 47: Przebiegi czasowe dla współpracy z 8086



Rysunek 48: Generacja sygnałów sterujących dla 8086



Rysunek 49: Obliczanie adresu

ku z tym, że zwykły strob zapisu pojawia się już po momencie próbkowania READY dlatego też sygnałem wyzwalającym logikę oczekiwania będzie sygnał K. Układ niegotowości można rozwiązać identycznie jak dla 68000 (P=K), na wejście S przerzutnika podana zostanie linia K[5].

[5]Składu dokonał LaTeX

Spis rysunków

Spis tabel