1. Dlaczego nie mozemy wprowadzic danych do komorki pamieci o adresie zero? Zrob zabezpieczenie zeby w razie gdy ktos chcial tam wprowadzić daną to przywoływany został system komputerowy ze specyfikacja zawarta w rejestrze A.

Programista nie może nic zapisać w komórce 0 pamięci, ponieważ zapamiętywany w niej jest adres powrotny podczas wywoływania przerwania.
[FORUM]
Zabezpieczenie przed wprowadzeniem danych do komórki pamięci o adresie 0(zero):
STA 0
C2q3t0: MAR <- MBR
C2q3t1: MBR <- A (chyba możemy to zrobić też wyżej, w jednym takcie), PAO start zapis
C2q3t2: "nic"
C2q3t3SV: (tutaj wywołujemy przerwanie) R <- 1, F <- 1
C2q3t3~SV: M <- MBR, F <- 0

ISZ
C2q6t0: MAR <- MBR, PAO start odczyt
C2q6t1~SV: MBR <- M
C2q6t2~SV: MBR <- MBR + 1, PAO start zapis
C2q6t3~SV: M <- MBR,**F <- 0** jezeli (MBR = 0), to (PC <- PC + 1)
C2q6t3SV: A <- MBR, R <- 1, F <- 1

SV - przerzutnik, flaga dla przywolania systemu operacyjnego poprzez przerwanie. Jest ona sprawdzana w cyklu wykonywania. Jest do niego wprowadzone wyjscie 16 wejściowej bramki NOR, do której to bramki wprowadzamy poszczegolne bity MBR.

2. Zaproponuj dodatkowe rozkazy dla procesora z wykładu i napisz dlaczego one są potrzebne jakos tak .

(X - liczba bezadresowych rozkazow z wykladow)
Rozkazy pozwalające na cykliczne przesunięcie:
CIR: q7 ~p c2 t3 MBR[X]: cir OA –przesuń cyklicznie w prawo O i A
CIL: q7 ~p c2 t3 MBR[X+1]: cil OA –przesuń cyklicznie w lewo O i A
A - akumulator
O - bit nadmiaru

Poszerzamy listę rozkazów o rozkazy pozwalające na obsługę wejść/wyjść:
OUT: C2 q7 t3 p MBR[X+2]: OUTR<-A[8-15], FGO<- 0 **{odczytaj znak z akumulatora}**
SKO: c2 q7 t3 p MBR[X+3]: Jeżeli (FGO = 1), to (PC <- PC + 1) {**omija jeśli jedynka w znaczniku wyjściowym }**

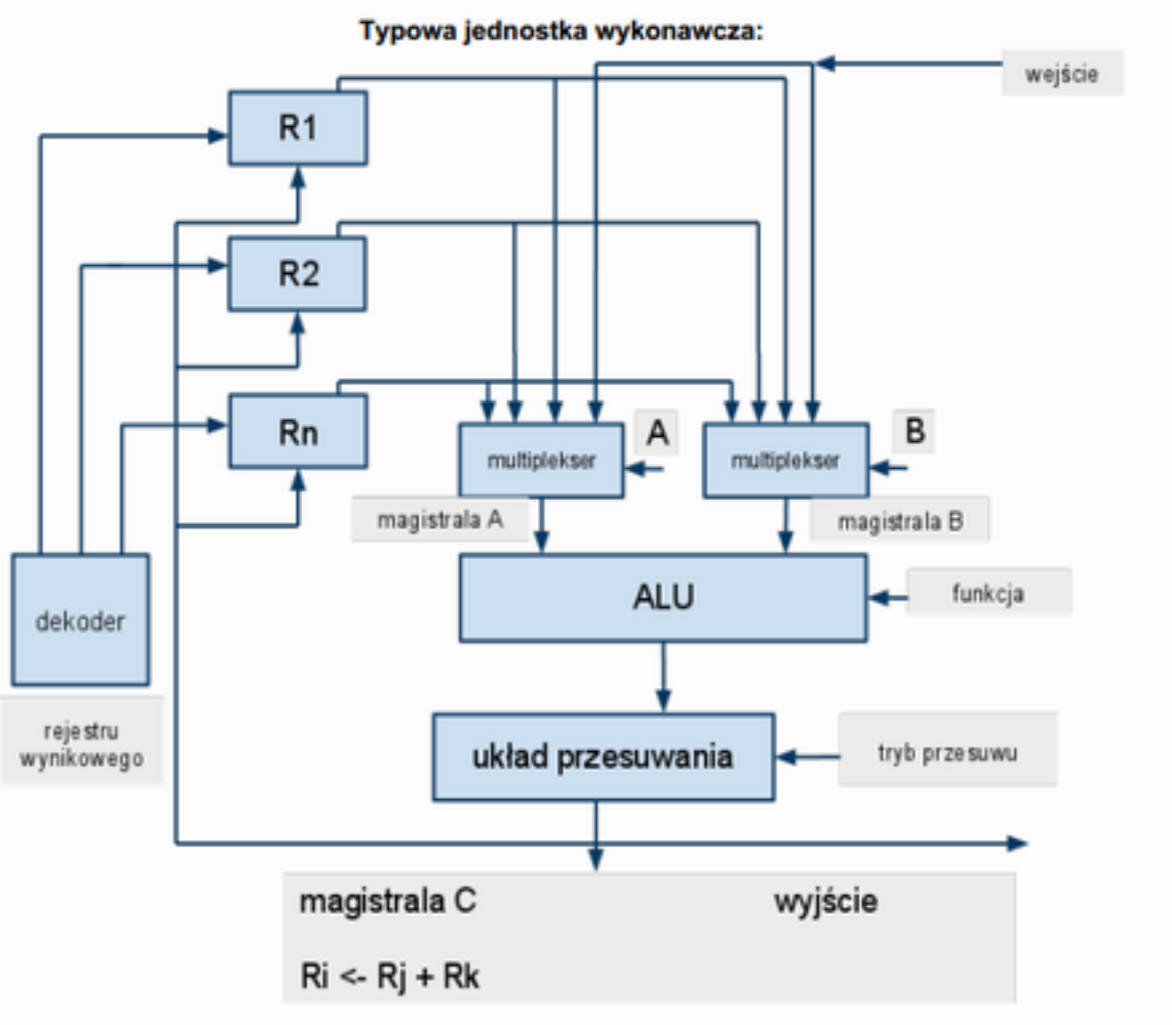
Dla założenia, że posiadmy wejście K odpowiedzialne za aktywacje całego układu, rozszerzamy listę rozkazów o:
HLT: q7 ~p c2 t3 MBR[X+4]: K<-0 - zatrzymywanie komputera, (chyba juz jest)

Schematy arbitrażu.

Daisy Chain (Stokrotka/Szeregowy) - wykorzystuje sie linie sterujaca przydzielajaca magistrale, przekazywana od urzadzenia o najwyzszym priorytecie do tego o najnizszym.
+ Prostota (tylko 3 druty);
- urzadzenie o najnizszym priorytecie moze nigdy nie uzyskac dostepu.
- wrazie gdyby nastąpiło uszkodzenie transmisji sygnalu GRANT, odciete sa pozostale urzadzenia
- GRANT ogranicza predkosc magistrali
- Linie release i request
Idenependent Request (Scentralizowany/Rownolegly) - kazde urzadzenie ma linie sterujaca zadajaca dostepu do magistrali, a centralny arbiter wybiera urzadzenie otrzymujace dostep. Uzywany w magistralach procesor-pamiec, szybkich urzadzeniach we/wy Uklad priorytetow moze byc programowany.
+ najszybszy, najbezpieczniejszy
- kosztowny
Arbitraz poprzez wzajemne wykluczanie - podobny do arbitrazu scentralizowanego, ale urzadzenia same okreslaja, które ma wyzszy priorytet. Jezeli magistrala jest wolna to urzadzenie chcace miec dostep do magistrali umieszcza swoj adres (kod) informujac pozostale urzadzenia o przejeciu uprawnienia 'master'. Obowiazuje zasada: kto pierwszy ten lepszy
losowanie (pooling) - Arbiter generuje adresy (kolejne losowo z priorytetem), urzadzenie chcace nadawac czeka na swoj adres wtedy staje sie 'masterem' i blokuje arbitra cechy: średniokosztowne, często używane
+ latwosc nadawania priorytetow
- zewn. scentr. Arbiter

Omówić układ sterujący i wykonawczy tego układu z wykładu

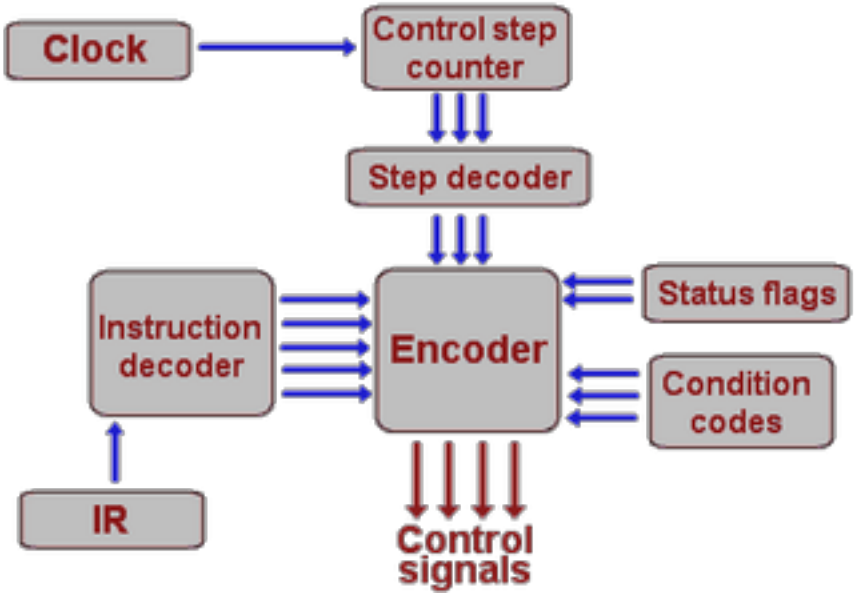
układ wykonawczy to ALU, rejestry, bufor i magistrala - jeśli źle to poprawdzie (moze znajdzie sie cos wiecej o tym?)
Typowa jednostka wykonawcza



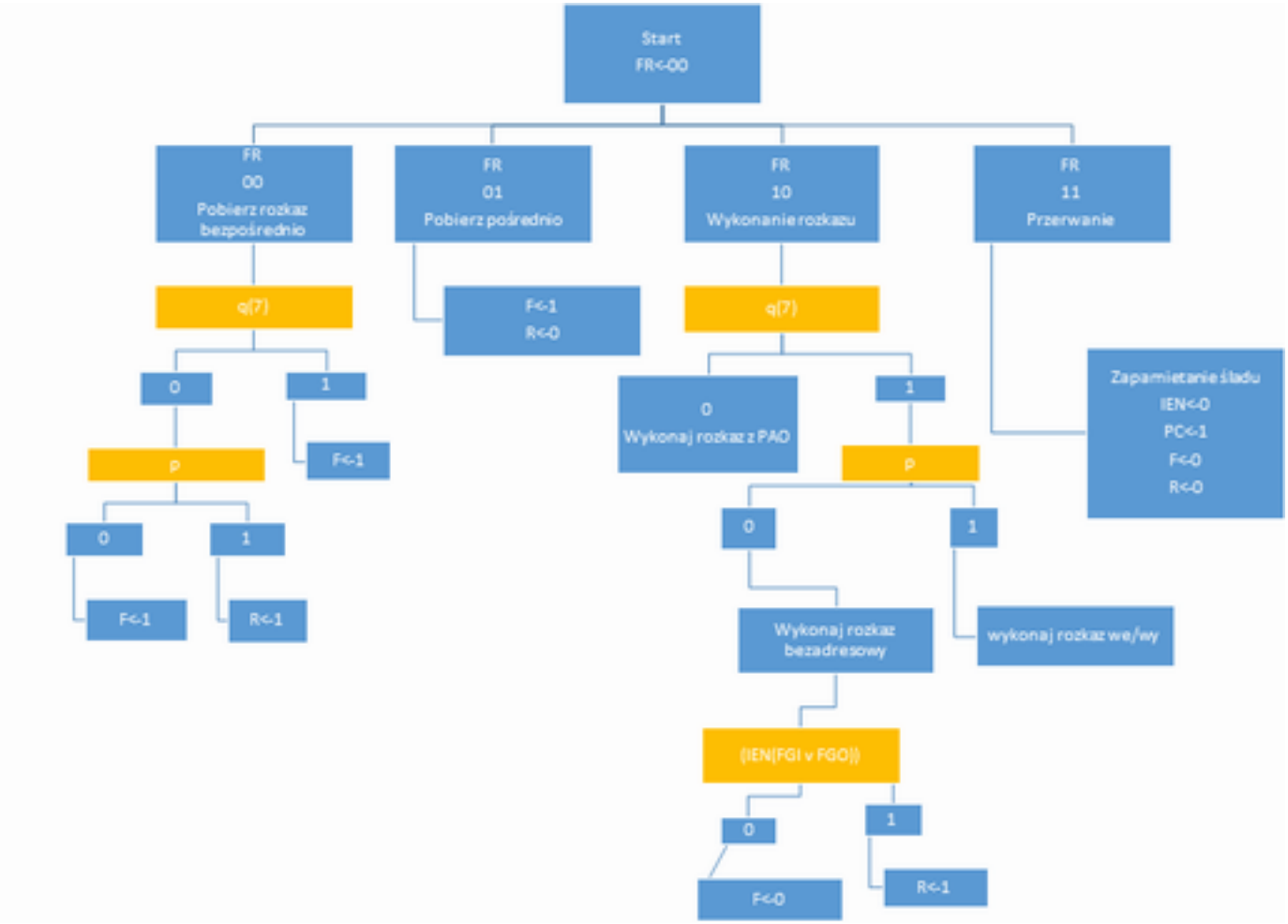
Typowa jednostka wykonawcza składa się z dekodera, rejestrów, układu arytmetyczno logicznego oraz z układu przesuwania. Dekoder wskazuje, do którego rejestru ma zostać zapisant wynik. ALU w zależności od zadanej funkcji wykonuje zadaną operację a następnie przesyła dane do układu przesuwania, który przesuwa je według ustalonego trybu.

W jednostce sterującej (CU) zawiera się rejestr rozkazów IR (instruction register), dekoder z układem sterowania (control), licznik rozkazów PC (program counter) i rejestr SR (status register). Rejestr rozkazów służy do przechowania kodu rozkazu pobranego z pamięci głównej (operacyjnej). W układzie dekodera, stosownie do treści pobranego kodu i bieżącego stanu procesora (pamiętanego w rejestrze stanu SR), wytwarzane są sygnały sterujące działaniem jednostek procesora i jego współpracą z pamięcią. automaty, sygnały sterujące, bity statusu, Jednostka sterująca jest „policjantem” czy też „kierującym ruchem” procesora. Monitoruje ona wykonywanie wszystkich instrukcji oraz transfer wszystkich informacji. Wydobywa instrukcje z pamięci, rozkodowuje je i dba o to, aby znalazły się w odpowiednim czasie we właściwym miejscu. Informuje też jednostkę ALU, których rejestrów ma ona użyć, obsługuje przerwania oraz — w celu wykonania pożądanej operacji — włącza odpowiedni zestaw obwodów elektrycznych w ALU. Aby odnaleźć następną instrukcje, jaka należy wykonać, jednostka sterującą wykorzystuje rejestr licznika rozkazów, natomiast,

aby „wiedzieć” na bieżąco o przepełnieniach, przesunięciach, „zapożyczeniach”, itp. — używa rejestru stanu."

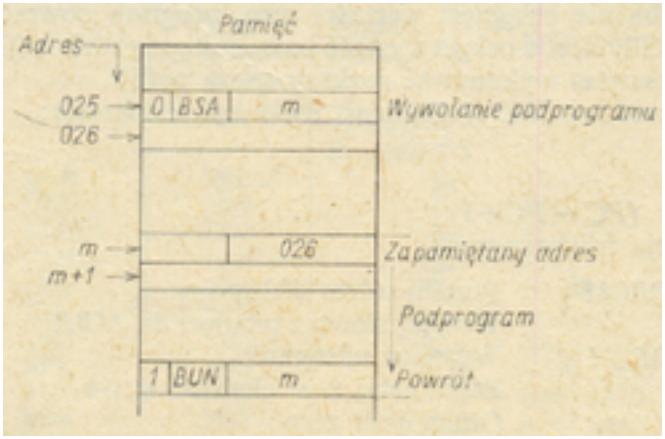


Powyżej typowa jednostka sterująca (żeby nie było że mamy schemat tej z wykładu)



Skok ze śladem - opis oraz opis w przypadku gdy jest dostępny stos

Skok ze śladem to skok do podprogramu z zapamiętaniem adresu z którego wychodził, aby mógł wrócić w to samo miejsce. W czasie wykonywania rozkazu adres następnego rozkazu znajdujący się w liczniku rozkazów jest zapamiętany w komórce pamięci określonej przez adres efektywny m. Wartość m+1 zostaje przesłana do rejestru PC i służy jako adres rozkazu do następnego cyklu pobrania. Powrót z podprogramu wykonywany jest za pomocą skoku bezwarunkowego z adresowaniem pośrednim.



BSA[MM] / JS (wyklad)
C2q5t0: MAR <- MBR, MBR <- PC, PC <- MBR, PAO start zapis
C2q5t1: M <- MBR
C2q5t2: PC <- PC + 1

Za panem Morrisem:
"Niektóre komputery przechowują adres powrotu w pierwszym słowie obszaru podprogramu, niektóre - pod ustalonym adresem w pamięci ; inne przechowują go w jednym z rejestrów procesora; jeszcze inne na stosie w pamięci.(...) Wywołanie podprogramu jest realizowane za pomocą następujących mikrooperacji:

SP <- SP + 1
M[SP] <- PC
PC <- Adres efektywny .

Jeżeli bieżący program wywoła inny to nowy adres powrotu jest umieszczany na stosie i tak dalej. Rozkaz realizując powrót z ostatnio wywołanego podprogramu jest wykonywany przez następujące mikrooperacje:

PC <- M[SP]
SP < SP - 1 "

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Cykl pobrania rozkazu
- C0t0 : MAR <- PC, PAO start odczyt
- C0t1 : MBR <- M, PC <- PC + 1 {rozkaz w MBR}
- C0t2 : IR[0-2] <- MBR[0-2], IR[3] <- MBR[3], MBR[0-3] <- 0
- ~q7pC0t3 : R <- 1
- (q7 v ~p)C0t3 : F <- 1

Cykl pobrania adresu pośredniego
- C1t0 : MAR <- MBR, PAO start odczyt {adres adresu operandu}
- C1t1 : MBR <- M
- C1t2 : "nic"
- C1t3 : F <- 1, R <- 0 {zawsze wykonanie rozkazu}

Modyfikacja cyklu wykonania rozkazu (dla przerwania):
● **C2t3(IEN (FGI v FGO))** : R←1
● **C2t3~(IEN (FGI v FGO))** : F←0

Cykl Przerwania:
● **C3t0** MBR ← PC, PC←0
● **C3t1** MAR ← PC, PC←PC+1, PAO start zapis
● **C3t2** M←MBR, IEN←0
● **C3t3** F←0, R←0

<u>Rozkazy:</u>
q0 – ADD { A<- A+M }
- $C_2q_0t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_0t_1$: MBR <- M
- $C_2q_0t_2$: OA <- A + MBR

q1 – LDA { A <- M }
- $C_2q_1t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_1t_1$: MBR <- M, A <- 0
- $C_2q_1t_2$: A <- A + MBR

q2 – AND {A<-A&M}
- $C_2q_2t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_2t_1$: MBR <- M
- $C_2q_2t_2$: A <- A ^ MBR

q3 – STA {pamiętaj zawartość akumulatora}
- $C_2q_3t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_3t_1$: MBR <- A
- $C_2q_3t_2$: M <- MBR

q4 – BRA {PC <- adres, skok bezwarunkowy}
- $C_2q_4t_0$: PC <- MBR

q5 – JS {skok ze śladem}

- $C_2q_5t_0$: MAR <- MBR, MBR <- PC, PC <- MBR, PAO start zapis
- $C_2q_5t_1$: M <- MBR
- $C_2q_5t_2$: PC <- PC + 1

q6 – ISZ {wyznacz następnik i skocz jeśli 0}

- $C_2q_6t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_6t_1$: MBR <- M
- $C_2q_6t_2$: MBR <- MBR + 1, PAO start zapis
- $C_2q_6t_3$: M <- MBR, gdy MBR = 0 to PC <- PC + 1

q7 – Bezadresowe:

1110 – ogólnego przeznaczenia (odwołania do rejestru). W rozkazach tych używa się pozostałych 12 bitów kodu rozkazu do określenia jednej z 12 różnych mikrooperacji.

CLA {zeruj akumulator}

$C_2q_7t_3$~pMBR[X]: A<-0

CLO {zeruj bit przeniesienia}

$C_2q_7t_3$~pMBR[X+1]: O<-0

NEGA { wyznacz dopełnienie zawartości akumulatora }

$C_2q_7t_3$~pMBR[X+2]: A<-~A

NEGO { wyznacz dopełnienie bitu przeniesienia }

$C_2q_7t_3$~pMBR[X+3]: O<-~O

CIR { przesuń cyklicznie w prawo O i A }

$C_2q_7t_3$~pMBR[X+4]: cir OA

CIL { przesuń cyklicznie w lewo O i A }

$C_2q_7t_3$~pMBR[X+5]: cil OA

HLT { zatrzymywanie komputera }

$C_2q_7t_3$~pMBR[X+6]: K<-0

1111 – rozkazy we/wy są potrzebne do przesyłania informacji do i z rejestru akumulatora, badania bitów znaczników i do sterowania przerzutnika zezwolenia na przerwanie.

INP { wprowadź znak do akumulatora }

$C_2q_7t_3$pMBR[X+7]: A[8-15] <- INPR, FGI <- 0

OUT { odczytaj znak z akumulatora }

$C_2q_7t_3$pMBR[X+8]: OUTR<-A[8-15], FGO<- 0

SKI { omija jeśli jedynka w znaczniku wejściowym }

$C_2q_7t_3$pMBR[X+9]: Jeżeli (FGI = 1), to (PC <- PC + 1)

SKO { omija jeśli jedynka w znaczniku wyjściowym }

$C_2q_7t_3$pMBR[X+10]: Jeżeli (FGO = 1), to (PC <- PC + 1)

ION { włącz przerwania }

$C_2q_7t_3$pMBR[X+11]: IEN<-1

IOF { wyłącz przerwania }

$C_2q_7t_3$pMBR[X+12]: IEN<-0

STOS

restart:

SP = 0, FULL = 0, EMPTY = 1

push:
SP <- SP + 1
SM(SP) <- MBR
jezeli (SP = 0), to (FULL <- 1)
EMPTY <- 0
pop:
MBR <- SM(SP)
SP <- SP - 1
jezeli (SP = 0) to (EMPTY <- 1) - instrukcja warunkowa ma jezeli a nie gdy (kazal to poprawiac z wykladow)
FULL <- 0

Dodajemy osobna jednostke pamieci dla stosu o wielkosci np 64 slow, tworzymy rejestr SP(6 bitowy), 1 bitowe rejestry EM,FL, poczatek w SP=0
PUSH:
- **c2t0~fl**: MBR<-A; SP<- SP+1
- **C2t1~fl**:STOS[sp]<- MBR
- **c2t2~fl:**Jeżeli SP=0 to fl<-1
- **c2t3:** EM<-0

Jeżeli [IEN(FGI V FGO)]=1 to R<-1
=0 to F<- 0

POP
- **C2t0 ~EM**: MBR<-STOS[SP]
- **C2t1~EM**: SP<- -SP-1
- **C2t2 ~EM**: jezeli SP=0 to EM<-1
- **C2t3** FL<-0

Jeżeli [IEN(FGI V FGO)]=1 to R<-1
=0 to F<- 0

Typy przerwań
- **Asynchroniczne** -> zewnętrzne: we/wy, zegar czasu rzeczywistego, zasilanie. (poza procesorem)
- **Synchroniczne** -> wewnętrzne: pułapki, wyjątki(exceptions) (wewnątrz procesora)
- **Programowe** -> SV calls (przywołania systemu operacyjnego).

Semantyka - języka programowania definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję w programie. Semantykę najczęściej definiuje się słownie, ponieważ większość z jej zagadnień jest trudna lub wręcz niemożliwa do ujęcia w jakikolwiek formalizm. Część błędów semantycznych można wychwycić już w momencie wstępnego przetwarzania kodu programu, np. próbę odwołania się do nieistniejącej funkcji, lecz inne mogą ujawnić się dopiero w trakcie wykonywania.

Pamięć - sklada się z pamięci operacyjnej + pamieci wirtualnej. Pamięć operacyjna jest podzielona na bloki(dł. bloku=dł. strony), a pamięć wirtualna na strony. Podczas adresowania

pamięci wydaje się że rzeczywista ilosc pamieci jest wieksza, jednak część danych znajduje sie w pamieci pomocniczej(np. na dysku twardym). Poprzez operacje przesyłania stron, potrzebne dane z dysku wędrują w stronach do pamięci operacyjnej. Odwzorowanie adresu rzeczywistego wymaga tylko adres strony(adres wiersza jest niezmienny, bo ma długość słowa). Pobieraniem ston zajmuje sie system op.

MMU-Zadania realizowane przez moduł zarządzający pamięcią:
● Translacja adresów logicznych na adresy pamięci fizycznej,
● Ochrona pamięci, kontrola uprawnień,
● Obsługa pamięci podręcznej,
● Zarządzanie szynami danych, przełączanie banków pamięci (mikrokontrolery 8-bitowe),
● Ułatwienie przełączania kontekstu (pamięć wykorzystywana przez proces, stos, itd...),
● Procesor ma bezpośredni dostęp do szybkiej pamięci statycznej, rejestrów oraz urządzeń mapowanych na pamięć,
● Każdy proces pracuje w oddzielnej przestrzeni pamięci.

Pamięć podręczna (cache):
Jest to bufor między pamięcią operacyjną, a rejestrami procesora. Jest ona niezbędna aby zniwelować dysproporcję pomiędzy wydajnością procesora a pamięci. Jest to bardzo szybka pamięć. Dane pobierane przez procesor najpierw są dostarczane do pamięci podręcznej z RAM, a następnie do procesora.

Adresowania:
Adresowanie względne służy do adresowania pamięci względem adresu aktualnie wykonywanej instrukcji w pamięci programu. Adres ten przechowywany w specjalnie do tego przeznaczonym rejestrze zwanym wskaźnikiem instrukcji (IP). Wykorzystuje się je przy rozkazach skoków.
Adresowanie bezpośrednie ma zastosowanie w instrukcjach wielobajtowych, w których w kodzie rozkazu następuje adres argumentu umieszczonego w postaci danych.
Adresowanie pośrednie – część kodu instrukcji wskazuje na komórkę pamięci zawierającą adres efektywny.

Bity statusu:
Bity znajdujące się w rejestrze stanu, ustawiane są na wyjściu ALU. Uzupełniają informacje zawracane, jako wynik ALU. C – przeniesienie (1 – przeniesienie wyjściowe ALU jest równe 1, 0 – jeśli przeniesienie wyjściowe równe 0), Z – bit wyniku 0 (1 – jeśli w wyniku są same 0, 0 – jeśli w wyniku jest chociaż jeden bit w stanie wysokim), S – bit znaku (najbardziej znaczący bit wyniku ALU), V – bit nadmiaru w U2.

TLB - jest to bufor translacji adresów stron w postaci asocjacyjnej pamięci podręcznej. Posiada stałą liczbę wpisów i służy do szybkiego odwzorowywania adresów logicznych pamięci wirtualnej na adresy pamięci fizycznej w komputerach stosujących stronicowanie pamięci. Stosuje się go w celu zredukowania dodatkowego obciążenia czasowego względem bezpośredniego dostępu do pamięci w zwykłej translacji. Kluczem na podstawie którego

lokalizowana jest pozycja w buforze TLB jest numer strony, a wartością na wyjściu jest numer ramki.