Organizacja i architektura komputerów [1]

Wykład 3

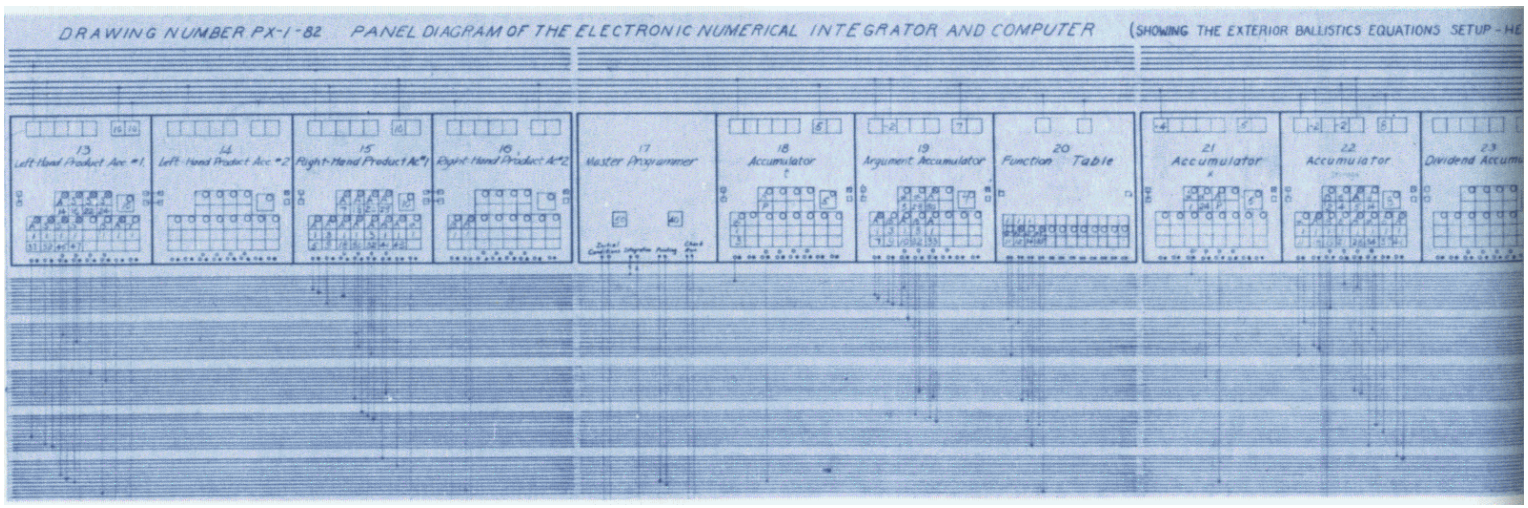
Piotr Patronik

8 marca 2016

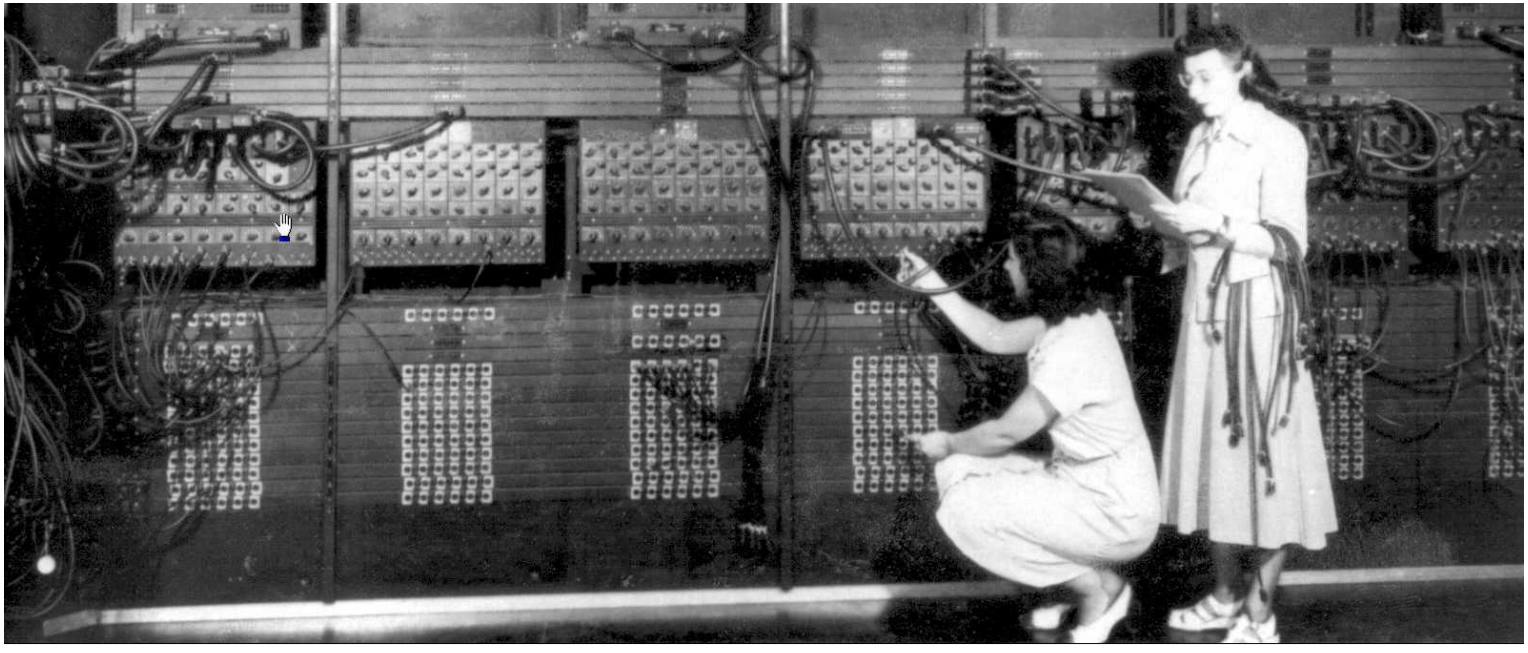
[1] (Prawie) dokładna kopia slajdów dr hab inż. J. Biernata

Program ENIACa

- Obliczanie tabel nastaw do rakiet balistycznych
- Symulacje fizyczne dla bomby wodorowej



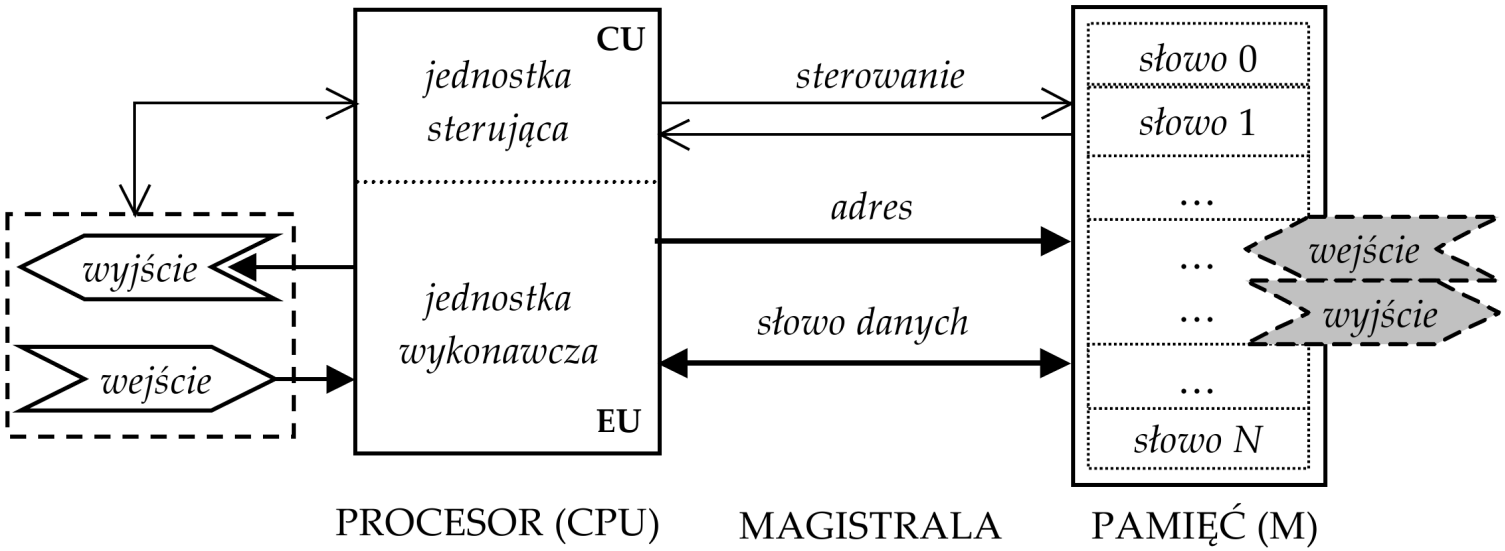
Programowanie ENIACa



- Zbiór ALU i plików rejestrów
- Ręczne zestawianie sieci połączeń

Komputer z programem przechowywanym (pamiętanym)

- program-stored computer (J. von Neumann, A. Turing, 1945)



? jak działa komputer?
... $\Rightarrow$ (**CU** : słowo: M$\rightarrow$CPU) $\Rightarrow$ (interpretacja **CU** : *polecenie*) $\Rightarrow$ (**CU** : słowo: M$\rightarrow$CPU) $\Rightarrow$ (interpretacja **CU** : *argument*) $\Rightarrow$ ... $\Rightarrow$ (wykonanie **EU** : wynik$\rightarrow$R) $\Rightarrow$ (interpretacja **CU** : słowo R$\rightarrow$M) $\Rightarrow$ (**CU** : słowo: M$\rightarrow$CPU) ...

? *gdzie jest umieszczone kolejne słowo?*

- pierwotna koncepcja: pętla poprzez modyfikacje rozkazów

Koncepcja pamięci

- Abstrakcja
  - Pamięć (S) – uporządkowany zbiór informacji jednostkowych (słów) s_i, $S = \{s_i; i \in Z\}$, $i = I, I+1, I+2 \ldots$
    - każde słowo ma ustalone miejsce w zbiorze – można mu jednoznacznie przypisać *numer porządkowy* (i) – unikatowy *wskaźnik lokacji* (adres)
    - słowa nie mają *etykiet* wskazujących ich znaczenie (brak typowania)
    - *rozmiar* każdego słowa jest taki sam
    - *interpretacja słowa* (określenie jego treści) zależy tylko od stanu maszyny w chwili jego pobierania z pamięci – zadanie *procesora*
- Realizacja
  - Pamięć – zbiór komórek (memory cell) – główny magazyn informacji
    - każda komórka pamięci zawiera jedno słowo (jednostkę informacji)
    - zawartość komórki pamięci (treść słowa) może zmienić tylko procesor dokonując (w wyniku wykonania rozkazu) przesłania słowa do pamięci

Nawigacja – gdzie są potrzebne słowa?

Sterowanie wykonaniem programu

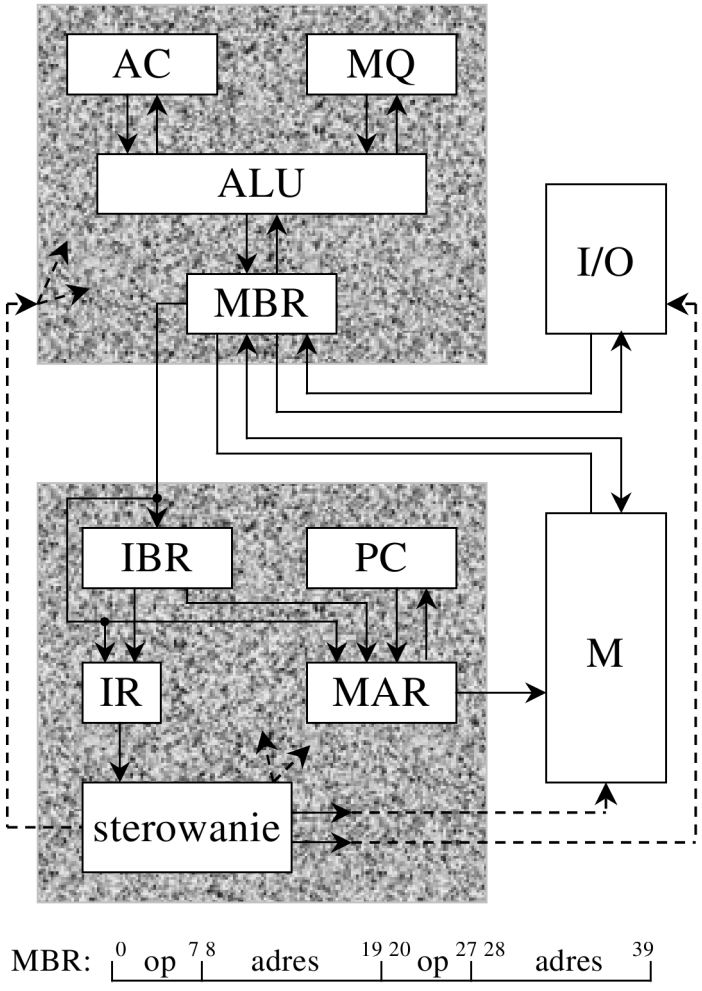
- tryb *control flow* – polecenie (rozkaz) czeka na dane
  - najpierm musi być pobrane słowo polecenia
    - lokacja pierwszego pobieranego słowa (punkt wejścia, pierwszy adres itp.)
  - z interpretacji słowa wynika zapotrzebowanie na argumenty
  - kolejne słowo polecenia musi być wskazane:
    - w treści polecenia bieżącego
    - automatycznie $\longrightarrow$ *licznik rozkazów* (*program counter*)
- tryb *data flow* – dane czeka*ją* na polecenie
  - osobne strumienie danych i poleceń
  - zestawy poleceń tworzą sekwencję opisaną przez algorytm
    - możliwe przetwarzanie współbieżne
  - zestawy danych są (częściowo) uporządkowane
    - możliwe jednoczesne przetwarzanie danych przez kilka rozkazów
  - brak komercyjnych realizacji *komputera* (GPU?)
  - in-memory processing, in-cache processing
  - powszechne realizacje jednostek wykonawczych (ALU)

Nawigacja – (udany) kompromis

- Adres (lokacja) kolejnego słowa polecenia
  - wskazana automatycznie (*licznik rozkazów*) – powiązanie *sekwencyjne*
    - wyklucza sterowanie – kolejne polecenie nie zależy od wyniku poleceń wcześniejszych
    - oszczędne – w kodzie polecenia nie jest potrzebny adres kolejnego polecenia
  - wskazana w treści polecenia bieżącego – powiązanie *łańcuchowe*
    - o rozwiązanie elastyczne, sterowanie dowolne
    - niepotrzebna rozbudowa kodu większości poleceń
- Kompromis
  - zasada – powiązanie *sekwencyjne* $\longrightarrow$ *licznik rozkazów*
  - wyjątek – powiązanie *łańcuchowe* $\longrightarrow$ możliwość sterowania
    - specjalne rozkazy skoku (jump)/rozgałęzienia (branch) $\longrightarrow$ wymuszanie stanu *licznika rozkazów*

Architektura komputera IAS (1952)

- Von Neumann → IBM 701



Lista rozkazów IAS (oprócz I/O)

STOR M(X)	$M(X) \leftarrow AC$
STOR M(X,8:19)	$M(X_{8:19}) \leftarrow AC_{0:11}$
STOR M(X,28:39)	$M(X_{28:39}) \leftarrow AC_{0:11}$
LOAD MQ	$AC \leftarrow MQ$
LOAD MQ, M(X)	$MQ \leftarrow M(X)$
LOAD M(X)	$AC \leftarrow M(X)$
LOAD –M(X)	$AC \leftarrow -M(X)$
LOAD |M(X)|	$AC \leftarrow \|M(X)\|$
LOAD –|M(X)|	$AC \leftarrow -\|M(X)\|$
JUMP M(X,0:19)	$IBR \leftarrow M(X_{0:19})$
JUMP M(X,20:39)	$IBR \leftarrow M(X_{20:39})$
JUMP +M(X,0:19)	if AC>0 then IBR←…
JUMP +M(X,20:39)	if AC>0 then IBR←…
ADD M(X)	$AC \leftarrow AC + M(X)$
ADD |M(X)|	$AC \leftarrow AC + \|M(X)\|$
SUB M(X)	$AC \leftarrow AC - M(X)$
SUB |M(X)|	$AC \leftarrow AC - \|M(X)\|$
MUL M(X)	$AC\|\|MQ \leftarrow MQ \cdot M(X)$
DIV M(X)	$MQ \leftarrow AC / M(X)$; $AC \leftarrow AC \bmod M(X)$
LSH (RSH)	$AC \leftarrow AC \cdot 2$ ($AC / 2$)

- (I)nstruction (B)ranch (R)egister, (M)emory (A)ddress (R)egister, (M)ultiplier (Q)uotient, (AC)cumulator . . .

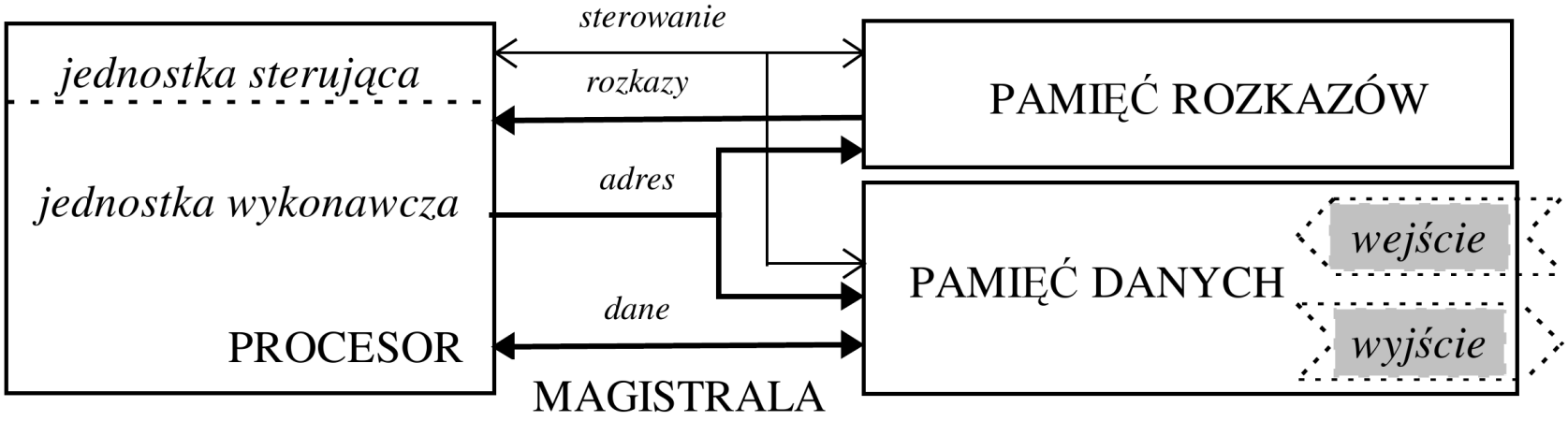
Lista rozkazów komputera EDSAC [Wilkes & Renwick, 1949]

		logiczne
C n	$AC \leftarrow AC \wedge M(n)$	Koniuguj AC z kodem $M(n)$ [*)]
L n	$AC \leftarrow AC \times 2^{M(n)}$	Przesuń arytmetycznie AC w lewo
R n	$AC \leftarrow AC \times 2^{-M(n)}$	Przesuń arytmetycznie AC w prawo
		arytmetyka stałoprzecinkowa
H n	$MQ \leftarrow M(n)$	Prześlij do rejestru mnożnika MQ liczbę $M(n)$
U n	$M(n) \leftarrow AC$	Przechowaj AC w $M(n)$
T n	$M(n) \leftarrow AC \ldots \leftarrow 0$	Przechowaj i wyzeruj AC
A n	$AC \leftarrow AC + M(n)$	Dodaj $M(n)$ do AC
S n	$AC \leftarrow AC - M(n)$	Odejmij $M(n)$ od AC
V n	$A \leftarrow A + AC \times M(n)$	Pomnóż dodając; $A = MQ \| \| AC$
N n	$A \leftarrow A - AC \times M(n)$	Pomnóż odejmując; $A = MQ \| \| AC$
X	$AC \leftarrow R(AC)$	Zaokrąglij AC
Y	$AC \leftarrow R(MQ \| \| AC)$	Zaokrąglij $MQ \| \| AC$
Z		Zatrzymaj i włącz dzwonek alarmowy
		rozgałęzienia
G n	$AC < 0 \Rightarrow$ GOTO n	Rozgałęzienie gdy $AC < 0$ (jeśli liczba w AC jest ujemna wykonaj rozkaz z komórki n, w przeciwnym razie wykonaj rozkaz następny)
E n	$AC \geq 0 \Rightarrow$ GOTO n	Rozgałęzienie gdy $AC \geq 0$ (jeśli liczba w AC jest dodatnia wykonaj rozkaz z komórki n, w przeciwnym razie wykonaj rozkaz następny)
		wejście / wyjście
I n	$M(n) \leftarrow$ TR	Odczytaj najbliższy rządek z taśmy perforowanej i umieść 5 tak pobranych bitów na najmniej znaczących pozycjach $M(n)$)
O n	TTY $\leftarrow M(n)$	Wyślij na dalekopis 5 najbardziej znaczących pozycji $M(n)$
F n	$M(n) \leftarrow$ PR	Skopiuj do pamięci kod wyprowadzany na drukarkę

[*)] AC – akumulator, $M(n)$ – zawartość komórki pamięci o adresie n

Architektura harwardzka

- H. Aiken, 1949

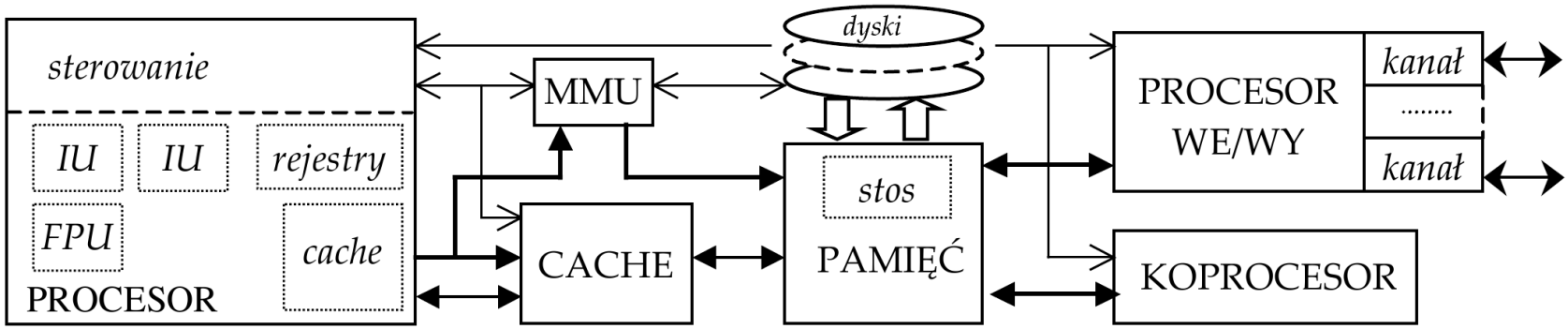


- osobne bloki pamięci danych i pamięci kodu (rozkazów)
- zalety
  - brak konfliktu dostępu do pamięci
  - ochrona kodu programu
- wady
  - etykietowanie danych
  - trudniejsza implementacja (dwie pamięci, dwa układy adresowe ...)

Modyfikacje architektury klasycznej

Niesprzeczne z modelem von Neumanna (*zachowana zasada działania*)

- rozbudowa systemu pamięci
- rozbudowa jednostki wykonawczej



- rejestry robocze (dużo)
- pamięć wirtualna i układ zarządzania MMU (*memory management unit*)
- pamięć masowa
- pamięć podręczna zewnętrzna (CACHE) i wewnętrzna (*cache*)
- kilka jednostek wykonawczych
- wspomaganie specyficznych działań – koprocesory

Działania na danych (1)

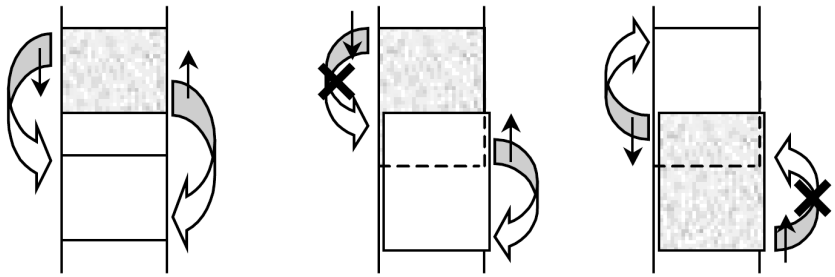
Klasyfikacja działań i fazy ich wykonania (format = struktura)

Działania na danych (2)

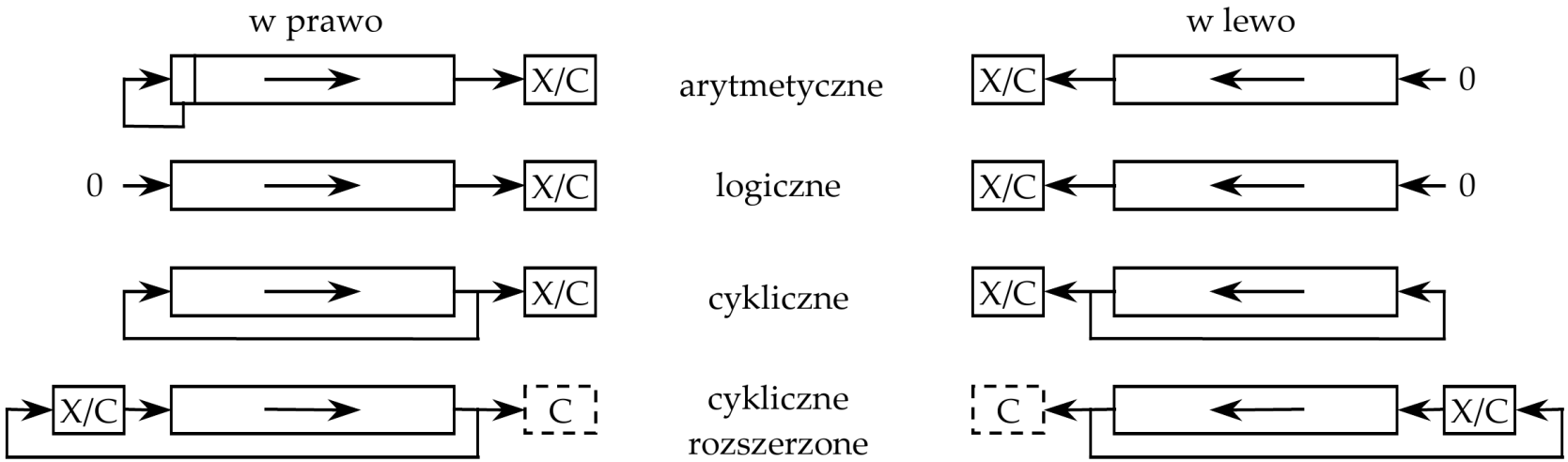
- *kopiowanie*
  - niszczące, nieodwracalne
  - kopiowanie bloków – ryzyko zniszczenia źródła (kolejność przesłań)
  - wymiana (*exchange*), (także przestawianie (*swap*)) – odwracalne
- *zmiana formatu*
  - przemieszczenie pól (rekordów) – przestawienie (*swap*)
  - przemieszczenie cykliczne bitów – rotacje i przesunięcia
  - rozszerzanie kodów liczb (sign/zero extend)
  - konwersje formatów zmiennoprzecinkowych
- *zmiana kodu*
  - upakowanie i rozpakowanie kodu
  - konwersje formatów liczb (zmienno/stało-przecinkowy)
  - przekodowanie (tablica przekodowań)
  - (negowanie słów)

Schematy działań (1)

- Transfer bloku słów zależnie od lokalizacji źródłowej (zacienione) i docelowej

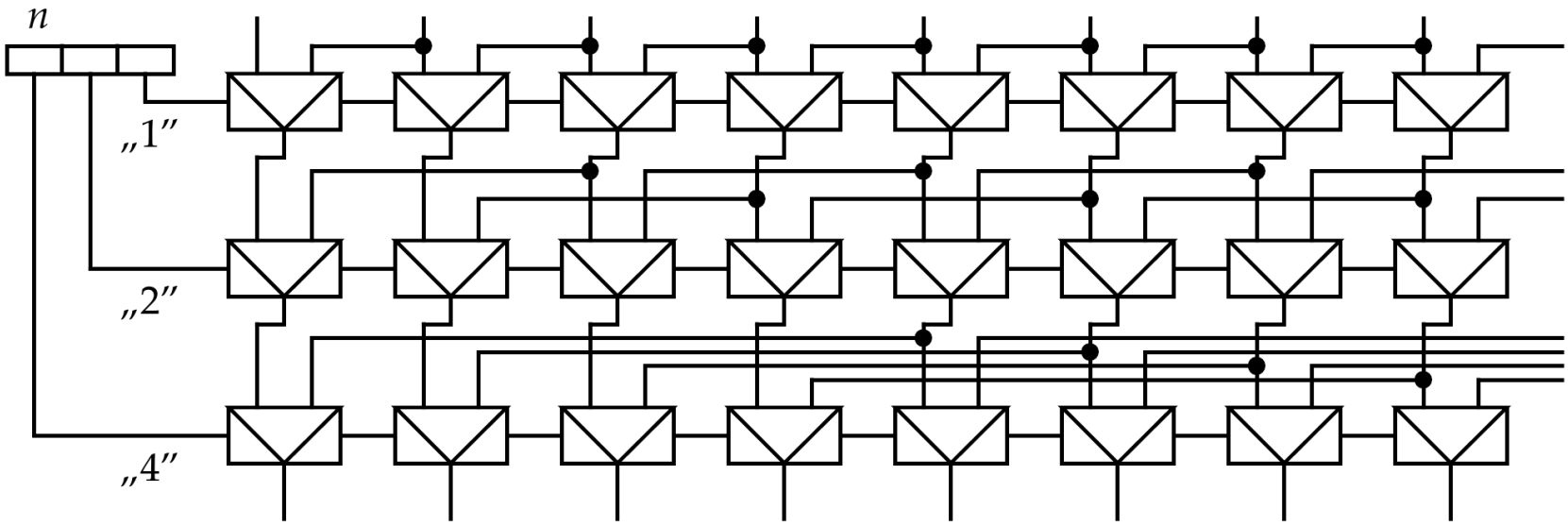


- Schematy przesunięć i rotacji (przesunięć cyklicznych)

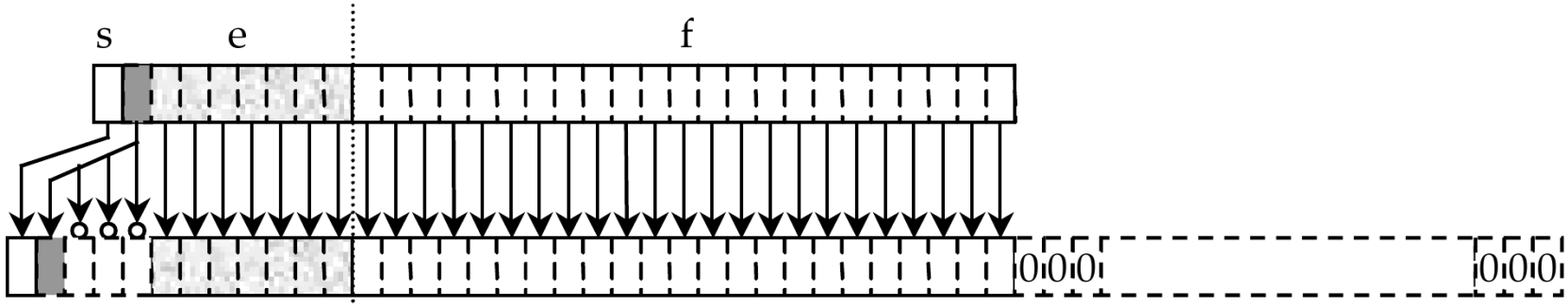


Schematy działań (2)

- Kombinacyjny układ
  przesunięć (w lewo) zbudowany z multiplekserów (*barrel shifter*)



- Konwersja formatu zmiennoprzecinkowego (o – negowanie bitu)



Mnemoniczny opis działań na poziomie architektury listy rozkazów (ISA)

Działanie	x86/Pentium Intel	MC680x0 Motorola	R2000 MIPS	PowerPC 601
Kopiowanie	mov, xchg, bswap	move, clr, exg, swap	move, mf, mto, lbu, lhu, lwu, la sb, sh, sw	lwz, lfs,… stw, stfs,…
skoki *	jmp, call, j*cc*, loop	jmp, jsr, bra, b*cc*, db*cc*	j, b, b*cc*	b, ba, bc, bclr, bcctr
Zmiana formatu	shl, shr, sar, rol, ror, rcl, rcr	lsl, lsr, asl, asr, rol, ror, roxr, roxl	sll, sra, srl, rol, ror,	sle, slw, sre, srw, rlmi, rrib
Zmiana kodu	movsx, movzx, xlat, aad, aam,	sext, zext	lb, lh, lw	extsb, extsh lbz, lhz,…
Działania logiczne	and, or, xor, not, test, set*cc*	and, or, eor, not, tas, s*cc*	and, or, xor, nor, not, s*cc*	and, nand, or, nor, xor, eqv, cmpl
Działania arytmetyczne	add, adc, sub, sbb, mul, imul, div, idiv, neg, cmp, daa, das	add, adc, sub, sbb, muls, mulu, divs, divu, neg, negx, cmp, abcd, sbcd, nbcd	add, addu, sub, subu, mul, mulo, mult, div, divu, rem, neg, negu	add, addc, adde, sub, subc, sube, mul, mullwu, div, divw, neg, abs, nabs, cmp

Konwencje opisu działań

składnia

- języki programowania:
  $\longrightarrow$wyrażenie
  wynik= *argument-a* DZIAŁANIE *argument-b* DZIAŁANIE ...
  $\longrightarrow$wywołanie funkcji
  wynik= FUNKCJA(*argument-a*, *argument-b* ...)
- asembler (poziom maszyny rzeczywistej):
  MNEMONIK *argument-a*, *argument-b*, ...

wskazywanie argumentów

- języki programowania:
  - *zmienne, stałe (obiekty)*
- asembler
  - *słowa w pamięci, rejestry procesora, stałe*

Model programowy procesora

wykaz rejestrów i ich cechy

- rejestry ogólnego przeznaczenia *GPR* (*general purpose registers*)
- rejestry specyficzne (niespójność architektury)

tryby adresowania

- sposoby tworzenia adresu w pamięci (głównej)
- ograniczenia (niespójność architektury) specyfikacja działań
- sposób tworzenia wyniku i jego syndromów
  - zasady
  - odstępstwa od zasad (niespójności architektury)
- ograniczenia użycia argumentów
  - nakaz użycia
  - zakaz użycia
- interpretacja argumentów
  - interpretacja kodów liczb
  - sposób tworzenia i przekształcania stałych (rozszerzenia kodu)

Asembler

rejestry: RISC – rejestry numerowane (zwykle 16, 32 lub rzadziej 64)

- stałoprzecinkowe – GPR (na przykład r0, r1, ...)
- zmiennoprzecinkowe – FPR (na przykład fr0, fr1, ...)
- specyficzne – sterowanie, zliczanie, powiązania wątków

CISC – rejestry nazywane:
MC68x00

- dane d0, . . . , d7 – 8/16/32-bitowe (#.b/ #.w/ #.d)
- adresy a0, . . . , a7 – 16/32-bitowe
- specyficzne – sterowanie

konwencje domniemywania

- rejestry kodów warunkowych
- rejestry sterujące
- rejestry specyficzne

Asembler 80x86/Pentium

model programowy – rejestry

- a, b, c, d w wersjach:
  - 8-bitowych #h, #l (al, ah, bl, bh, cl, ch, dl, dh)
  - 16-bitowych #x (ax, bx, cx, dx)
  - 32-bitowych e#x (eax, ebx, ecx, edx)
  - 64-bitowych r#x, #l (rax, rbx, rcx, rdx)
  - (blokowe – mm0,...,mm7, xmm0,...xmm7)
- adresowe
  - 16-bitowe (si, di, bp, sp)
  - 32- bitowe (esi, edi, ebp, esp)
- segmentowe (adres bloku)
  - 16-bitowe (cs, ds, es, fs, gs, ss)
    $\rightarrow$specyficzne – flagi (kody warunkowe), sterowanie

konwencja domniemywania –

- rejestr akumulatora (al, ah, ax, eax, rax, dx:ax, edx:eax)
- zliczanie (cx, ecx, rcx)

Adresowanie 80x86/Pentium

adresowanie pamięci tryb adresowania – składowe adresu i sposób obliczania

- MC68x00 – (tryb adr.) np. (d7.w, a3)
- architektura RISC – [tryb adr] lub domniemanie (na podstawie działania)
- 80x86/Pentium – segment:[tryb adr.], np. ds:[eax+4*ebx+przemieszczenie]
  - podstawowy 8086 – dozwolone tylko zestawienia:
    - seg:[bp|sp+si|di+przemieszczenie]
      seg = cs, ds, es, ss
  - rozszerzony 80386 – jedno ograniczenie (baza $\neq$ esp):
    - seg:[baza+indeks+przemieszczenie]
      seg = cs, ds, es, fs, gs, ss
      baza|indeks = eax, ebx, ecx, edx, esi, edi, ebp, esp

Adresowanie 80x86/Pentium w składni AT&T

Dygresja

Asembler as (Dokumentacja do binutils, sekcja 9.15)

- `.intel_mnemonic` vs `.att_mnemonic`

Adresowanie pośrednie – składnia Intel vs AT&T

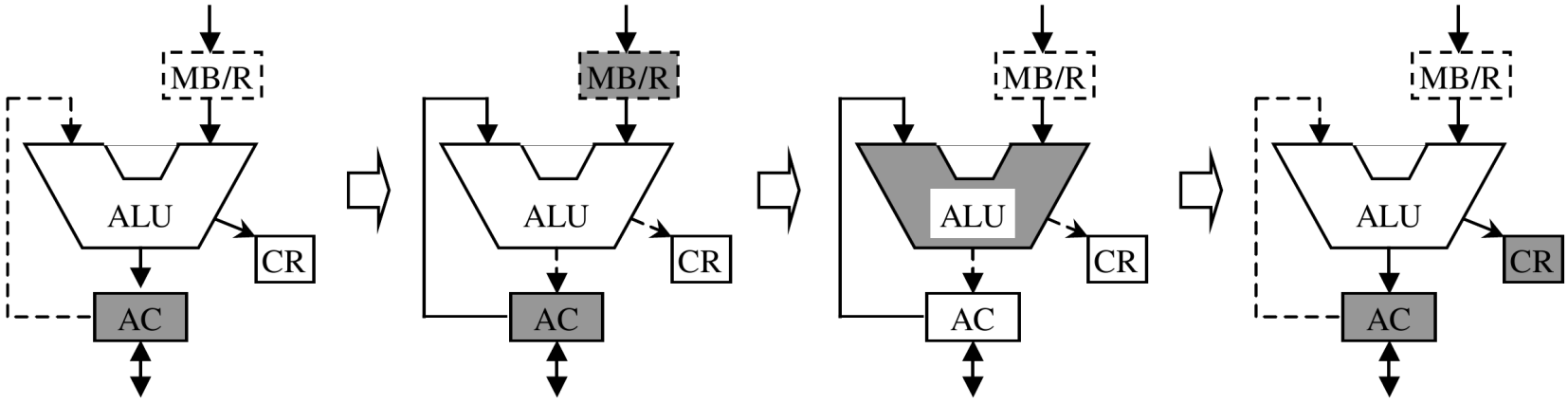
- `seg:[baza+indeks*skala+przemieszczenie]` (Intel)
- `seg:przemieszczenie(baza, indeks, skala)` (AT&T)

Przykłady

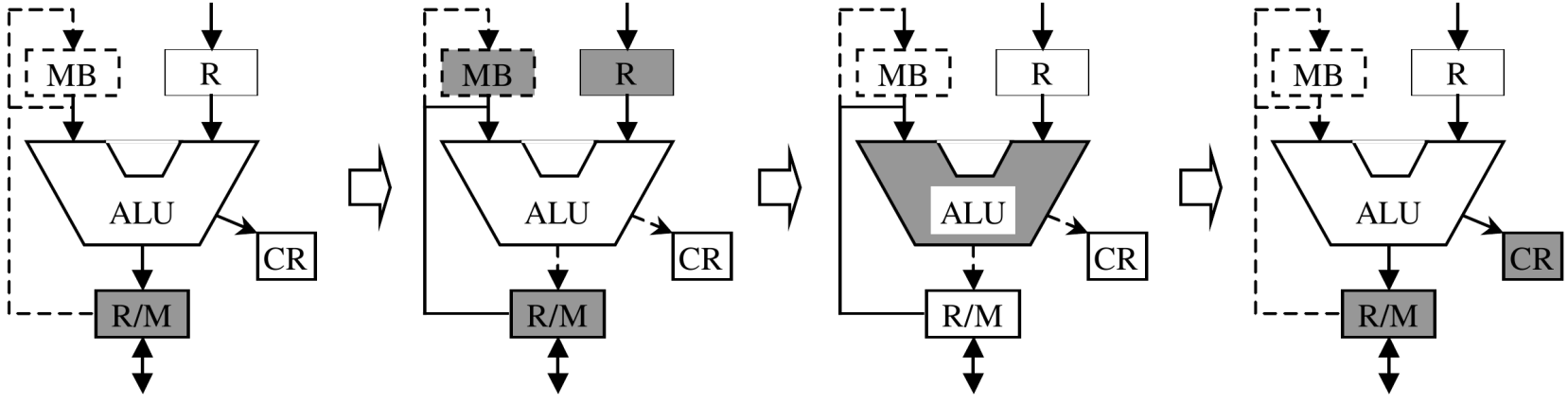
Intel	AT&T
`[ebp-4]`	`-4(%ebp)`
`[foo+eax*4]`	`foo(,%eax,4)`
`[foo]`	`foo(,1)`
`[foo]`	`foo`
`gs:foo`	`%gs:foo`
`foo`	`$foo`

Architektura jednostki wykonawczej procesora

- organizacja akumulatorowa i rozszerzona (AC $\rightarrow$ R)

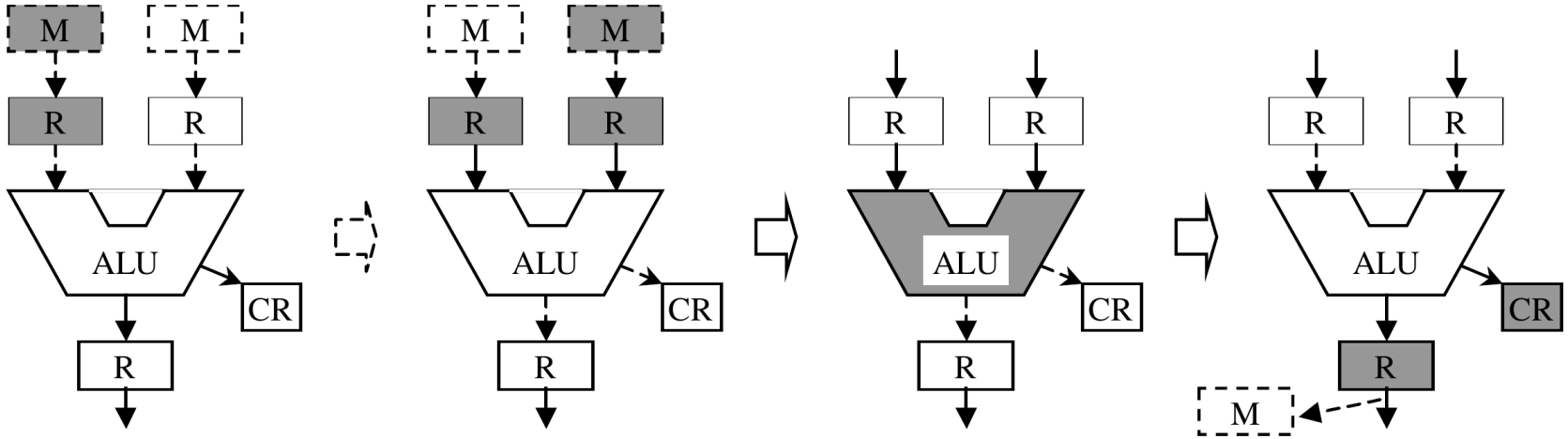


- organizacja akumulatorowa uogólniona (AC $\rightarrow$ M/R)



Rejestrowa i stosowa architektura jednostki wykonawczej

- organizacja rejestrowa (*load–store*) i uniwersalna (R → R/M)



- organizacja stosowa