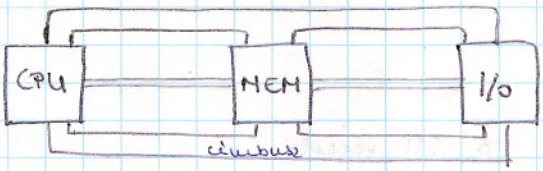
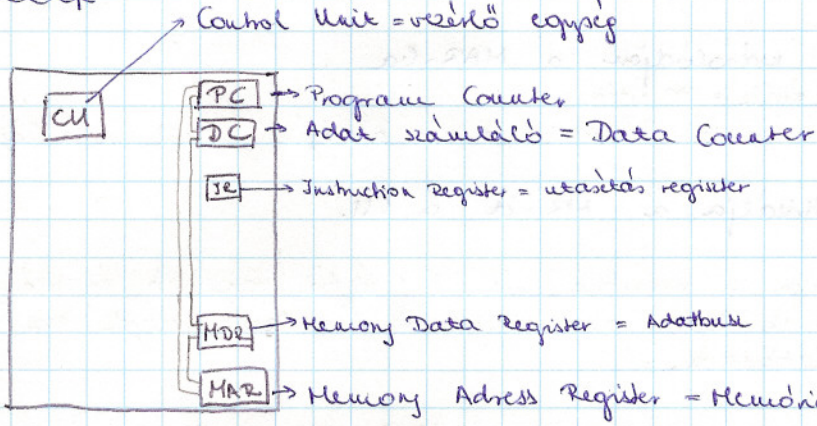


CPU belseje:



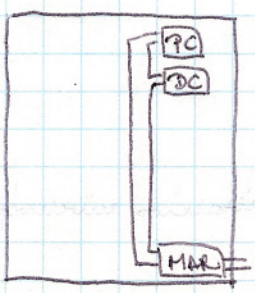
- PC: - program számláló
  - számokat ad elő → működési elve (000, 001 ... stb)
  - Neumann elméletében a programok egymás után következnek

PC: címtároló, megmutatja, hogy mi a köv. utasítás címe

- DC → adat helyének megmutatása végett tárolja a címet

címbusz: 16 db vezeték (CPU integrált áramkörből 16 láb)

- MAR: címek külső fizikai megjelenése, reprezentálása a chipeken (látszanak a címező lábak)
  ő maga a címbusz



- CU:- Ő irányít, hogy a MAR felvegye-e a PC értékét
- MDR:
- IR: eltárolja azt az utasítást, amelyet éppen végre kell hajtani
- A MR és MW műveleteket a CU vezérli

Példa: beolvassuk a köv. utasítást

① PC másolódjon a MAR-ba
MAR értéke = PC értéke
A memória felismeri, h a cím alapján melyik cellát az a bájt

② Aktiválja a MR-et a CU.

Tudja a memória, melyik bájtról van szó, és már azt is, hogy olvasni kell ⇒ megjeleníti majd az adatbuszon.

(Ha a memória címe A3 ⇒ MDR is A3-as lesz)

③ MDR másolódjon be az IR-be (az utasítás tárolójába)

④ PC+1=PC ⇒ megnöveli a PC értékét

A PC a köv utasítás címét tartalmazza

Ettől van szó a beolvasás ⇒ ez nem program, hanem a CPU működése. Ezt nem kell előírni, folytonosan végrehajtódik.

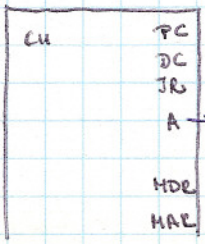
Végrehajtja az A3-nak megfelelő kódot a CPU és kezdi elölről.

Képzeljük el, hogy ez a folyamat az adatbeolvasás feladata.

① DC másolódjon a MAR-ba

② MR aktív

③ MDR → A



→ Akkumulátor → általános műveletvégző regiszter

Ha már nem nő az adatszámláló értéke eggyel.

↑ Ez egy adatkiolvasási folyamat.

Adatírás:

① DC → MAR — MAR-ba másolódik a cím

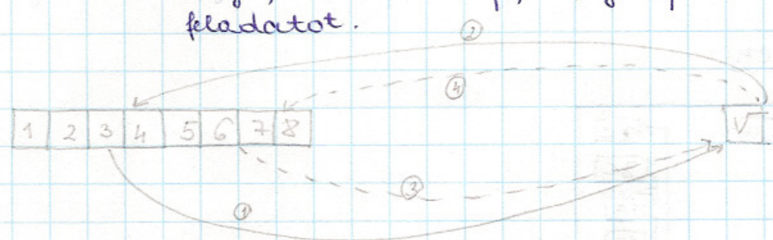
② A → MDR — MDR-be másolódjon az Akkumulátor ⇒ adatbusz = akkumulátor, címbusz = DC

③ HW aktív
↳ írás

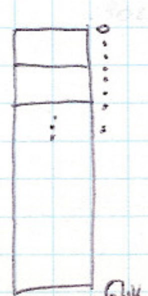
Feladat: Számoljuk ki több kifejezés gyökét. Kérdés: Az adatot tegyük be mindenhová?

A gyökszámolás folyamatát eldugjuk a memóriába.

Program működése: A memóriában eljut oda, hogy kellene a gyök, elugrik a számolási folyamathoz más helyre, ha kiszámolja, folytatja tovább a feladatot.



- Az elkérő utasítás után a következő helyen folytatja. Kellene egy hely, amely megjegyzi a PC értékét, majd ha megint kell, elő tudjuk venni. ⇒ Keresni kell egy helyet, ahol tárolni tudjuk a PC értékét.
- Az első processzorokban külön regisztert tartottak arra, hogy „el lehessen dugni" a PC címét. (16 bites)
- Miért nem 100%-ig kielégítő megoldás ez?

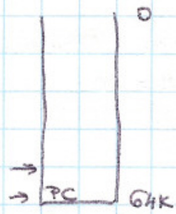


Az alsó részét biztos nem használjuk majd semmire ⇒ használjuk a mem végét arra, hogy ott tároljuk a PC értékét.

Ez a: stack v. verem v. zsák.

Ez a tárolási elve a LIFO. (A sor: FIFO eljárás)

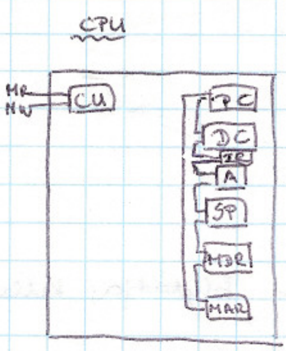
A tároló, amit mutat: u. mennyire van tele a memória alulról.



Az ilyen adatpakolásokat azért végezzük fordított sorrendben, mert így lehet memória, lehet adat.

Két művelete van: PUSH → betenni
POP → kivenni

SP → Stack pointer: megmutatja, „meddig" telt meg a memória



CALL → hívás

(Töltés esetén csökken a mutató! A 0-tól kezdődik)

1. PC → [SP] — az akt programszámlálót tegye oda, ahová a stackpointer a memóriába mutat
2. SP - 1 → SP
3. új érték → PC — PC új értéket kap

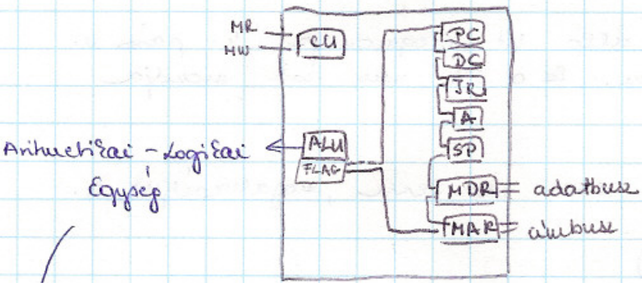
A CALL PÁRJA a RETURN. Ezek páros utasítások.

RET → visszatérés (ugyanaz a gyök. az érték az A??-ba kerül ⇒ vissza kell írni)

① SP + 1 → SP

② [SP] → PC   a stackpointer által mutatott érték kerül a programszámlálóba.

- A stack lehetőséget biztosít netintből netint hívni, CALL-ból lehet CALL-ba menni. → Ez a PC-k világában ez korlátlan helyet jelent.



→ Aritmetikai - logikai műveleteket végez.

FLAG: (zászló) - regiszter

- bitekből áll, de ezek nem olyan összefüggőek, mint a PC-nek. (ott a bitek egymástól függő értékeket jelentenek)
- itt mindegyik önálló jelentéssel bír, de a memória tudja egyben kezelni.

CARRY-bit (CY) → egyik proc-nál ez a legfelső bit, máshol a legalsó
→ az elvégzett aritmetikai művelet eredménye elfér-e 8 biten (ekkor 0 az érték), v. nem fér el (1-et ad.)

ZERO-bit (Z) → az elvégzett aritm-log. művelet eredménye 0 lett (1-et ad) v. nem lett 0.

pl.: 0+0 = 0
128 + 128 → 0 ⇒ CY = 1, Z = 1.

SIGN-bit (S) : → előjelbit
ha ⊖ ⇒ 1-et ad
ha ⊕ ⇒ 0-t ad

pl.: $-3-(+2)=-5$
pl.: $-3+8=5$

Van olyan (PC is ilyen), amely a 3 bitből 1-et készít.

Pl.: Ha kettes komplementet használ, és nem fér el 8 biten ⇒ túlcsordulás keletkezett.

OVERFLOW (O) : túlcsordulás

PARITY-bit (P) : → paritás-bit

→ az utolsó helyen álló bit megmondja, h. páros v. páratlan-e a szám. Ez a bit, nem ezt mondja meg.

→ megszámoljuk, hány 1-es van benne, megállapítjuk, h.
(8 → 1db; 7 → 3db)
páros v. páratlan az 1-esek darabszáma, és a 8 bit mellett ezzel azt jelezzük, h. a kommunikálásra kerülő bitsor páros v. páratlan.
ha párosat adunk át +1 ⇒ plen
ha páratlant -"- +0 ⇒ plen
(jelezhetnénk a fordítottját is, teljesen mindegy.)

→ A FLAG-regiszterben a paritás-bit egy állapotot jelez (h. páros v. páratlan)

→ haszna: program végrehajtása alatt a program sosem egyenes, mert a CALL-al „lépegetünk". Osztani akarunk, de 0-val nem lehet ⇒ mielőtt a programot lefuttatnánk el akarjuk dönteni, h. az osztó 0-e v. nem.
↓ HIBA ↓ SZÁMOLÁS

megvizsgáljuk, h. a zero-bit 0-e. Ha igen,

elugrunk egy helyre → kiírotás, vagy, ha 1 →

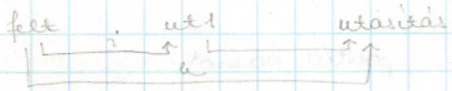
számolunk.

Elágazás → CARRY-bit

Feltételes elágazás kicsinyített fajtája: vagy máshová lépünk,

vagy folytatjuk.

(SKIP) kis ugrás (1 utasítást tud kihagyni
↳ ha 0 → ugrás, ha 1, nem csinál semmit.



a feltételes JUMP ugrás a jellemző

• A feltételes utasításnak is van 1 kódja.

Speciális ugrás → akármi a feltétel, mindenképp ugrani kell =
FELTÉTEL NÉLKÜLI UGRÁS (szélsőséges eset)

<u>A PC utasításai:</u>

① végrehajtás sorrendjét befolyásoló utasítások (programszámláló értékét befolyásoló utasítások)

② Aritmetikai, logikai → ÉS, VAGY, XOR (3 alapművelet)
↓
összeadás, kivonás
nem minden processzorban van szorzás és osztás

③ Léptető - forgató utasítás

1 helyiértékkel jobbra, 1 helyiértékkel balra