PROCESSZOROK I-II

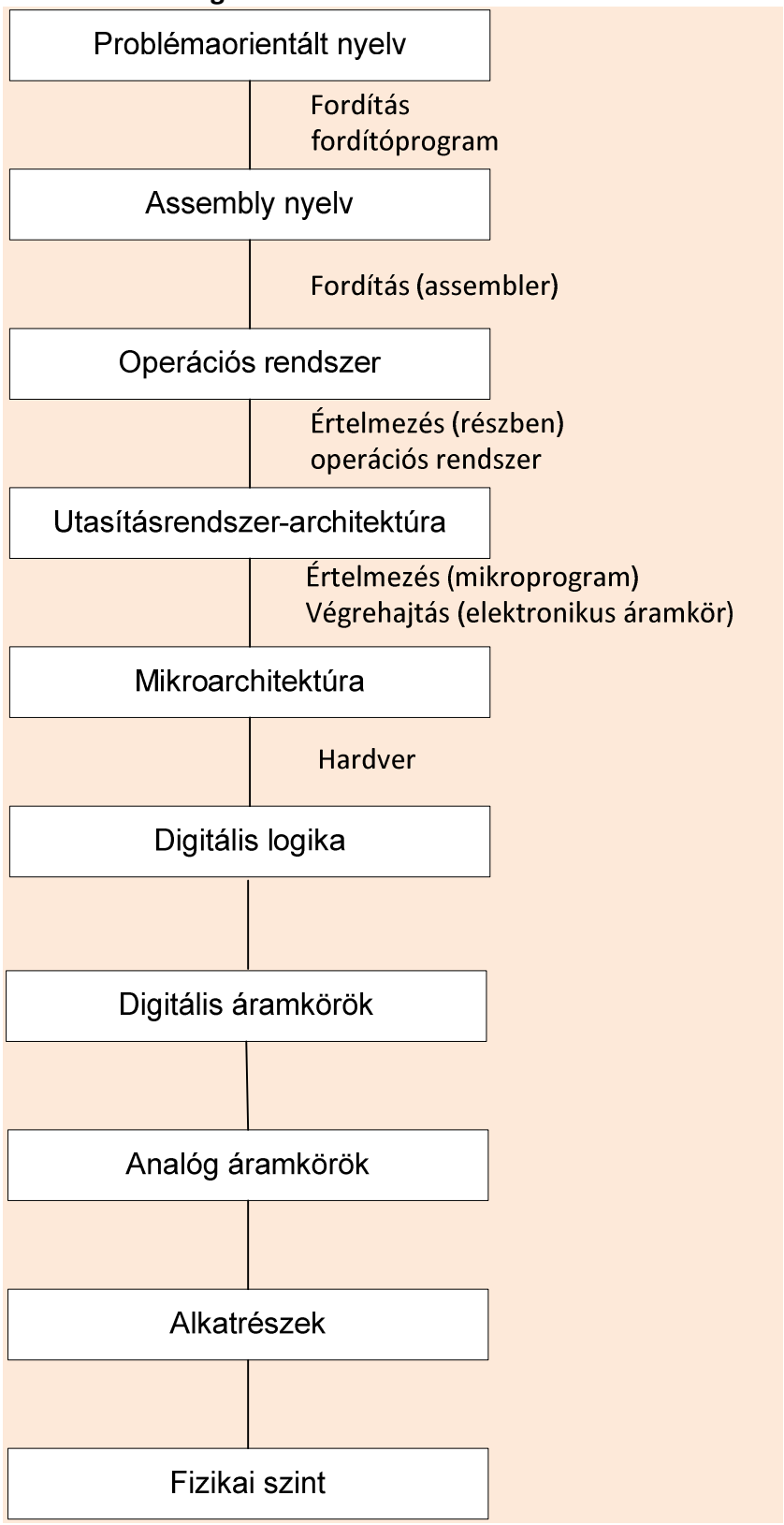
Tartalom

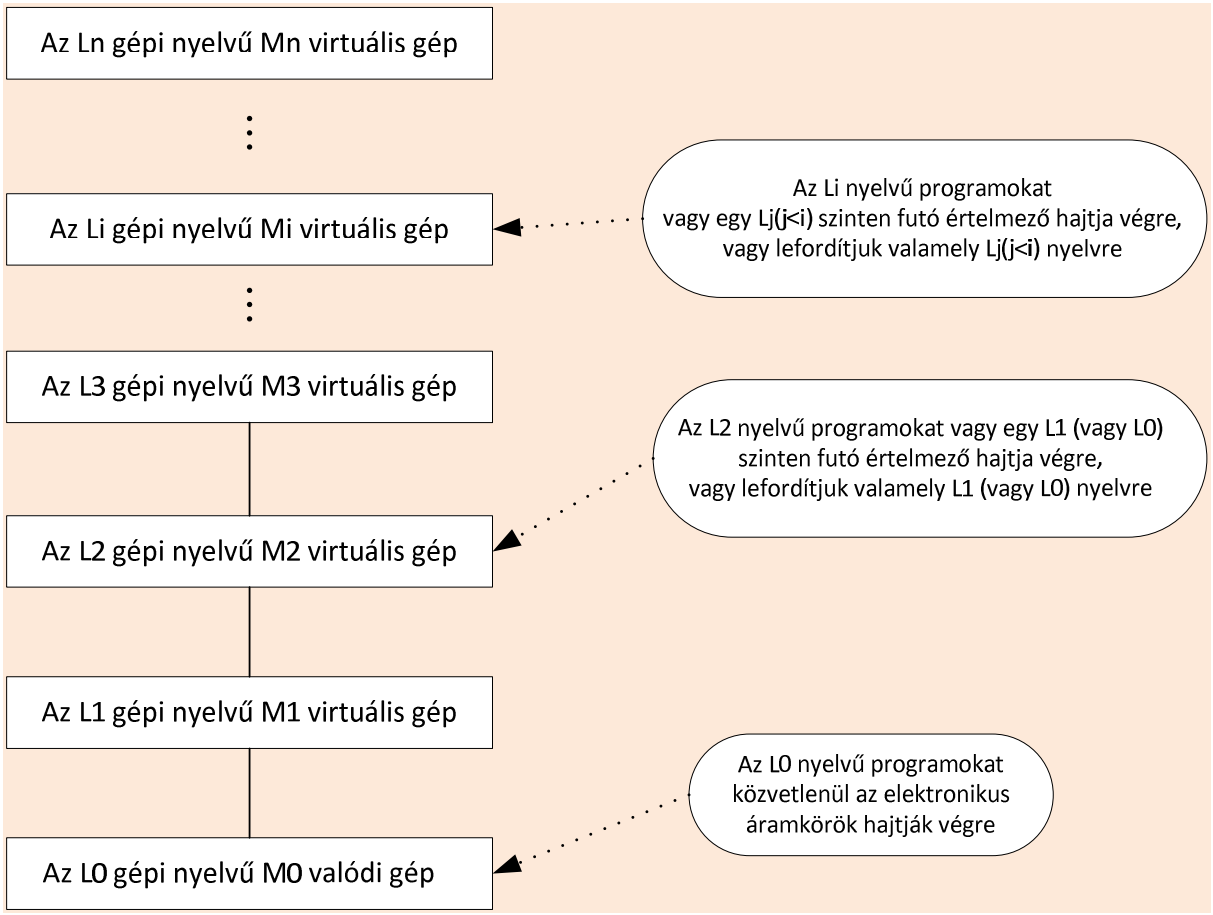
PROCESSZOR MODUL

01 rétegmodell Tanenbaum+ 01

(processzormodulhoz)

TOP-DOWN **BOTTOM-UP megközelítés**





(A számítógép rétegmodellje a Tanenbaum analógiájára készült.)

Az ábrán rétegek vannak:

legalul : L_0 gépi nyelvű réteg, amely a M_0 valódi gépen fut
a megértéshez a szemantikai rést az ember és a gépi kódú szint között át kell hidalni, ehhez különböző mennyiségű virtuális gépet használhatunk (az ábrán a különböző szintek

legfelül: L_n gépi nyelvű M_n gép

általánosan: egy valamely közbülső $L_{(i)}$ nyelvű programot valamely alacsonyabb szinten futó értelmező hajt végre
vagy lefordítjuk valamely alacsonyabb szinten levő nyelvre

a legalsó szinti, az L_0 nyelvű programokat már közvetlenül az elektronikus áramkörök hajtják végre.

02 rétegmodell Tanenbaum+ 02

A digitális számítógép az általunk neki adott utasítások alapján valamilyen algoritmikus problémát old meg.

Azoknak az utasításoknak az együttese, amit a számítógép megért, és amelyben az ember ki tudja adni ezeket az utasításokat, a *gépi nyelv*.
Az adott problémára vonatkozó utasítás sor, amelyben azt adjuk meg, hogy hogyan oldja meg a számítógép a feladatot, a *program*

Meg kell adnunk, hogy mit értünk értelmezésen:
értelmezés: ha veszünk valamely L szinten lévő programot, azt megfeleltetjük a legalsó szinti L_o-nak (valamilyen szintű utasítás sorozat), és azt azonnal végrehajtatjuk a számítógéppel

fordítás: az $L_{(i)}$ nyelven megírt program minden utasítását helyettesítjük az L_0 nyelv utasításainak sorozatával, és azokat hajtatjuk végre

Az ember számára nagyon nehézkes ezen az L_0 szinten kommunikálni, ezért ennek a szemantikai résnek az áthidalására szolgál a nyelvek és a számítógépes/virtuális nézetek hierarchiája

Az egyszerű gépi nyelv egyszerűbb elektronikát tesz lehetővé, viszont az ember-gép kommunikációt tehát megnehezíti

Ha a piramist lefordítjuk számítógépi szintekre, az ábrán látható:
legalsó: digitális logika
következő: mikroarchitektúra
következő: utasításrendszer architektúra
következő: operációs rendszer
következő: Assembly nyelv
legfelső: problémaorientált nyelv

A szintek száma növelhető: vannak szintek
a digitális logika *alatt*, és
a problémaorientált nyelv *felett*: itt az *alkalmazás* magasabb szintnek tekinthető

A fenti szintfelosztás rokon az adatátviteli rendszerek protokolljainak modellezésével

03 rétegmodell Tanenbaum+ 03.docx

Az ötödik szinttől kezdve, a problémaorientált nyelvek szintjén (pl. CC++, Java, stb.)(?!)

A következő, negyedik szint az Assambly nyelv szintje, ahol már szimbolikus leírással találkozunk.
Itt kell megjegyezni, hogy az ötödik, a problémaorientált nyelvek szintjén, tömören tudjuk leírni a feladatot, amit a számítógépnek végre kell hajtania, míg ahogyan a szinteken egyre haladunk lefelé, a tömörséget egyre inkább terjedelmesség váltja föl, ez egy piramisnak

megfelelő módon ábrázolható. Legtömörebbek a legmagasabb szinten vagyunk, a legkevésbé tömörek pedig az elektronikai szinten. Az Assembly nyelv szintje tehát szimbolikus nyelvi szint. Ennek a szintnek a felépülését a későbbiekben vizsgáljuk.
A harmadik szint az operációs rendszer szintje, ami a rendszer erőforrásaival való gazdálkodás szintje.
A második szint a gépi utasítások szintje, a továbbiakban részletesen fogjuk vizsgálni.
Az utolsó előtti szint a mikroarchitektúrák szintje. Ha a gép mikroprogramozott, ez a szint értelmezi a fölötte levő gépi utasítás szintet, és a regiszterek és az aritmetikai-logikai egység, az ALU segítségével hajtja végre, illetve a vezérlőegységben lévő mikroprogram vezérli az adatfolyamot vagy adatutat a végrehajtás során.
A legalsó szint a digitális logika szintje, ahol kapuk, illetve egy bites-több bites memória- és regiszter elemek vezérlése történik.

Rendben van a szintek sorszámozása?

04 rétegmodell Tanenbaum+04.docx

A rétegmodellben a rétegek között tudunk mozogni: például az Assembly nyelv szintjéről az Assembler fordítóprogram segítségével tudunk alsóbb szintre kerülni.
Elvileg mindenhol egységes vagy meghatározható a fordított irányú átlépés, például a disAssembler program segítségével.
A második szinten, tehát a gépi utasítás szintjén dől el a két gép kompatibilitása. Ebben a modellben erre támaszkodva az emuláció is értelmezhető.

05 rétegmodell 05

Az emuláció során egy adott gépen megírt programot futtatunk egy másik gépen, amelynek más a gépi kódja illetve a gépi utasításszintje. Ahhoz, hogy ezt a problémát megoldjuk, pl. az egyes vagy a kettes szint fölé beiktatunk egy olyan mikroprogramot, amely mikroprogramozott módon utánozza a másik gép utasításkészletét, így egy mikroprogram szintű szimulációt hajtunk végre.
Választhatjuk azt az utat is , hogy a gépi utasítás szintjén írunk egy olyan szimulációs programot, amely az adott valós gépet szimulálja, és lefordítja a másik gép gépi utasításaiban írt programot az adott gép gépi utasításainak megfelelően. Így a két gép között látszólag fordítás nélkül tudjuk hordozni a programokat.
A mikroprogramozott szint azonban gyorsabb verziót eredményezhet.

06 rétegmodell coo-design 06.docx

Amikor a tervezést végezzük, és például a hardver-szoftver codesign módszerét választjuk, valójában azt döntjük el, hogy a szemantikai rést milyen mértékben töltjük ki hardver szintekkel, és milyen mértékben szoftver szintekkel. Ez a modell akár egy szintű is lehet, tehát azonnal olyan elektronikus megvalósítást végzünk, ami nem igényli a további szintek meglétét. Ez tervezői döntés kérdése.

07 szgp felépítése – a huzalozott számítógép 07

ábra…..

Az ábrán látszik, hogy három csoportból áll a számítógép:

első:	a memória köré csoportosul	jelölés: M
második:	az ALU köré csoportosul	jelölés: ALU (*arithmetic-logic unit*)
harmadik:	vezérlés rész	

A memória rendelkezik egy címregiszterrel, ami megcímzi a benne lévő utasításokat és/vagy adatokat, ez az AR vagy *address register*.
A memória kimenetén van az adatregiszter vagy *data register* (DR), ide kerülnek a memóriából kiolvasott utasítások és adatok, mert itt Neumann típusú számítógépről beszélünk, azaz az adatok és utasítások vegyesen helyezkednek el a memóriában.
A címregiszter, az AR (*address* register) a PC-ről (*program counter*), azaz programszámlálóról kapja a bemenetét.
A programszámláló olyan regiszter, ami sorozatos végrehajtás esetén mindig mutatja a következő utasítás címét, azaz úgy viselkedik, mint egy pointer, és az implementálás műveletet értelmezni tudja és végre tudja hajtani saját magán.
Az ALU egyik bemenetén van az adat regiszter, a DR a memóriából, a másik bemenet az akkumulátor, az ACC regiszter, ami ennek a rendszernek a legbonyolultabb regiszter műveleteit végre hajtani képes regisztere.
Az adatregiszter másik elágazása vezet az IR-hez, az utasításregiszterhez, aminek a végrehajtása során a vzérlőegység kiadja a számítógép struktúrájának a megfelelő vezérlőjeleket.
Ebben a felépítésben az IR (*instruction register*) kimenetén a PR, azaz *fase* (fázis) regiszter található ezek kapcsolódnak a vezérlés blokkhoz.

08 szgp felépítése – elemek – huzalozott – sínes struktúra 08

Az előző részben leírt felépítésben a legfontosabb elemek a regiszterek, a memória, az ALU és a vezérlés voltak.
A regiszterekben szavakat tárolunk, amelyeket a hosszukkal lehet jellemezni: egy n bites regiszterben n bit hosszúságú adatot tudunk tárolni.
A regiszter műveleteket képes végrehajtani a vezérlés segítségével a benne tárolt adatokon.
Az aritmetikai-logikai egység végrehajtja azokat a műveleteket, amelyeket a számítógép utasításkészlete határoz meg, a vezérlés pedig szolgáltatja ehhez a megfelelő jelek sorozatát.
A memória, ahogy azt már korábban meghatároztuk, az adatokat és az utasításokat tárolja.
Ebben a felépítésben az elemek között közvetlen kapcsolatot létesítünk, ezt nevezhetjük huzalozott kapcsolatnak.
Ha figyelembe vesszük azt, hogy időben szétválasztott műveletek sorozatáról van szó, akkor megelégedhetünk azzal, hogy az adatátvitelhez ezek az elemek közös eszközöket használjanak, hiszen ezek nem egyidőben szükségesek, illetve hogyha az adók és a vevők

viszonyát tekintjük, akkor ha egy adó és több vevő kapcsolódik egy ilyen eszközhöz, a megoldás kielégítő lehet számunkra.
Így jutunk el a sínes struktúrához, ami szervezettebb, könnyebben áttekinthető szerkezetet eredményez, a huzalozott megoldás előnye pedig az, hogy gyorsabbak lehetünk.
A sínes struktúrát úgy is elképzelhetjük, hogy például az előbbi számítógép felépítésben megfogjuk a memóriát és az összes kapcsolatát, mint valami gumiszalagot képzeljük el, és ha ezt kihúzzuk a szerkezetből, struktúrából valahová oldalra, akkor látjuk, hogy az AR-hez és a DR-hez kötődő része fog így megnyúlni, ezek szolgálnak majd az adat- és a cím sínhez a memória elérésekor. Ugyanez megtehető lenne az input/output egységekkel is, hogyha ezen a struktúrán a teljes Neumann architektúra szerepelne.
Ha a vezérlésekre is alkalmazzuk ezt az elvet, eljuthatunk a három sínes számítógép felépítéshez, ami a mikroprocesszorok felépítésének is egyik szokásos formája.

08 1 szgp felépítése elemek és műveletek 081

Elemek: memória, regiszterek, ALU, vezérlés

A regiszter egy tárolóeszköz, egy szó vagy adategység tárolására alkalmas. Jellemezhető a hosszával.
A regiszteren különböző műveletek is értelmezhetők és megvalósíthatóak.
A regiszterbe lehet írni, az adatokat ki lehet olvasni párhuzamosan és /vagy sorosan, lehet a regiszter tartalmát léptetni jobbra-balra, ennek változatait a gépi utasításoknál majd részletesen kifejtjük, valamint lehet a regiszter tartalmát törölni, ez azt jelenti hogy a teljes regisztertartalom 0-ba állítódik.
Ezeken a műveleteken kívül a regiszterekkel értelmezhetők és megvalósíthatók egyéb műveletek is: különleges léptetések, aritmetikai-logikai, ciklikus gyűrűs léptetés, stb. A programszámláló (PC) regiszternél lehetséges a regiszter tartalmának aritmetikai inkrementálása. Ehhez a regiszter rendelkezik a megfelelő logikával.

Ebből a szempontból a memória sokkal egyszerűbb: cellákból áll, ez azért lényeges, mert ebből a szempontból mindegy, hogy adatot vagy utasítást tartalmaz. Ez abból dönthető el, hogy milyen utasítás és milyen módon fog az adott cellához vagy memóriarekeszhez fordulni. A memóriarekeszek írhatók és olvashatók: a memória szempontjából ezek az értelmezhető műveletek.
Az aritmetikai-logikai egység egy olyan hálózat, ami a gépi utasítások által meghatározott műveleteket képes végrehajtani a vezérlő egység (*control unit*, CU) irányítása alatt. Többféle változata van, és a gépi utasításoknak megfelelő műveleteket tudja elvégezni.

A vezérlő egység úgy is tekinthető, mint a gépi utasítások értelmezője. A fázis regiszterbe vagy a mikroprogramtár címregiszterbe kerül az utasítás kódja, ezt értelmezi a vezérlő egység, és ennek megfelelően generálja a vezérlő jel sorozatot.

08_2 szgp felépítése -- belső sínek - bus 082.docx

09 A 02 gépi utasításkészlet teljessége.docx

A fogalmat az algoritmus és architektúra meghatározásra építjük.
Egyszerűen úgy fogalmazhatunk, hogy a gépi utasítás teljessége azt kívánja, hogy az alkalmas legyen a megvalósítani kívánt algoritmusok leírására az adott architektúrával.
Kicsit kiterjesztve ezt a meghatározást az emuláció esetére, figyelembe véve a nyelvek és rétegek modelljében megadott értelmezést, hozzájön egy olyan követelmény, hogy az utasításkészlet lefordítható legyen a fizikai gép nyelvére.

10 01 IF EX

Az utasítások végrehajtása

Az utasítások végrehajtása két alapvető részből áll:

1. az utasítás lehívása, angolul *fetch* (*instruction fetch: IF*)
2. az utasítás végrehajtása, angolul (*instruction) execution* (*EX*)

Mind a lehívás, mind a végrehajtás további belső lépésekkel finomítható.
A lehívási rész belső felépítése az utasításokra egyforma, míg a végrehajtási rész az utasítások típusától függően különböző lehet.

10 02 IF+, EX+

Az utasításlehívási rész (a fetch-rész) minden esetben tartalmazza a szétválasztást, ami az utasítás műveleti kódra és címzésre vonatkozó részre történő szétválasztását jelenti, az utasítás dekódolását. A szétválasztásnál a megfelelő regiszterekbe, tehát az *instruction register*be, az utasítás regiszterbe például betöltjük az utasítás műveleti kód részét, végrehajtjuk a dekódolást, és elindul a végrehajtás. A fetch részhez tartozik az utasításszámláló 1-gyel történő növelése is, hacsak nem elméleti négy címes utasításról beszélünk.
Az utasítás végrehajtása például egy ilyen elméleti négy címes utasítás esetén nagyon egyszerű, hiszen az utasítás tartalmazza a két operandus címét, ezeket le kell hívnunk a memóriákból, tehát az *execution*/végrehajtás fázisban az utasítások végrehajtása következik, majd a művelet elvégzése után következik az eredmény visszaírása arra a címre, ami szintén az utasításban szerepel. Ebben az esetben ott rendelkezésünkre áll a következő utasítás címe is, azaz a fatch-nél jelölt programszámláló 1-gyel történő növelése az ilyen címzési rendszer, az ilyen utasítás felépítés esetén nem szükséges. Ha egy címes utasításról van szó, ami műveleti kódból és címzésmód részből, valamint *displacement*-ből, vagy címrészből áll, abban előbb el kell végeznünk a címszámítási részt – tehát itt az execution-ben a címszámítás következik -, miután ez megtörtént, lehívhatjuk az operandust, hiszen a másik operandusunk rendelkezésre áll mondjuk az akkumulátorban, elvégezzük a műveletet, és itt is visszaírás következik. Ez egy egyszerű Neumann-elvű gép általános utasításvégrehajtására vonatkozik.

Magára az utasításvégrehajtásra majd még a Harvard architektúra és a CISC és RISC, elsősorban a RISC processzor működése kapcsán visszatérünk.

Mikroprocesszoros digitális rendszer elemei
Mikroprocesszorok felépítése és működése
Mikroprocesszor: utasítás-készlet, I/O kezelés, megszakítások, DMA

Alapfogalmak

Rendszer, digitális rendszer
Véges Automata (FSM = Finite State Machine), Mealy és Moore automata
Vezérlők, programozott vezérlők

Digitális rendszer funkcionális elemekkel

Digitális rendszerek: számítógép, mikroszámítógép, mikroprocesszor

Mikroprocesszor = CPU – egy tokban
Mikrokontroller = Mikroprocesszor + MEM + I/O – egy tokban
Mikroszámítógép = Mikroprocesszor vagy Mikrokontroller + MEM + I/O – egy kártyán (SBC = Single Board Computer)

Digitális rendszerek mikroprocesszorral, mikrokontrollerrel és mikroszámítógéppel

Mai nagy számítógépek: Számítógép = szuper-mikroprocesszor (egy tokban) +MEM + I/O

Kiindulás – a korai számítógépek: Számítógép = CPU + MEM + I/O

Neumann elvek, Neumann Architektúra (SISD)

Forrás: Horváth L

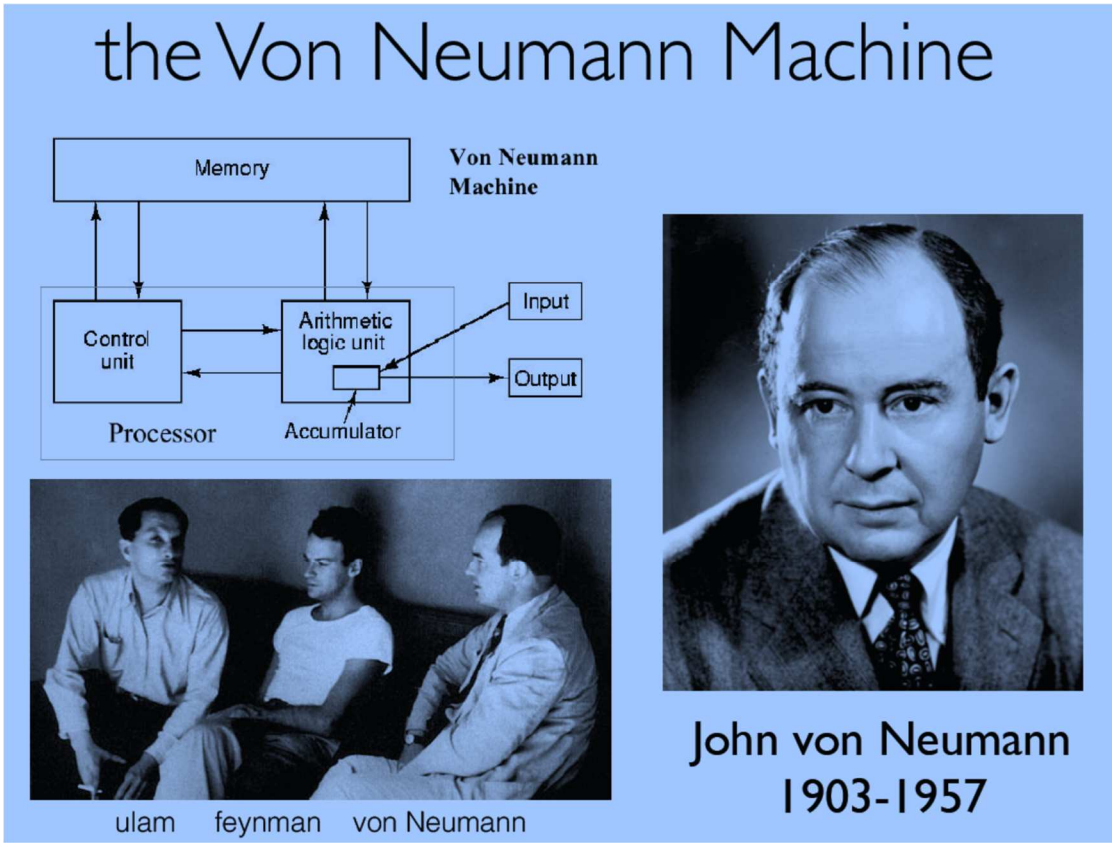
1945. júniusában a philadelphiai egyetem Moore Intézetében a mechanikus lyukasztó másológépeken megoldott feladatok tapasztalataiból kiindulva Neumann a következőket javasolta:[5]

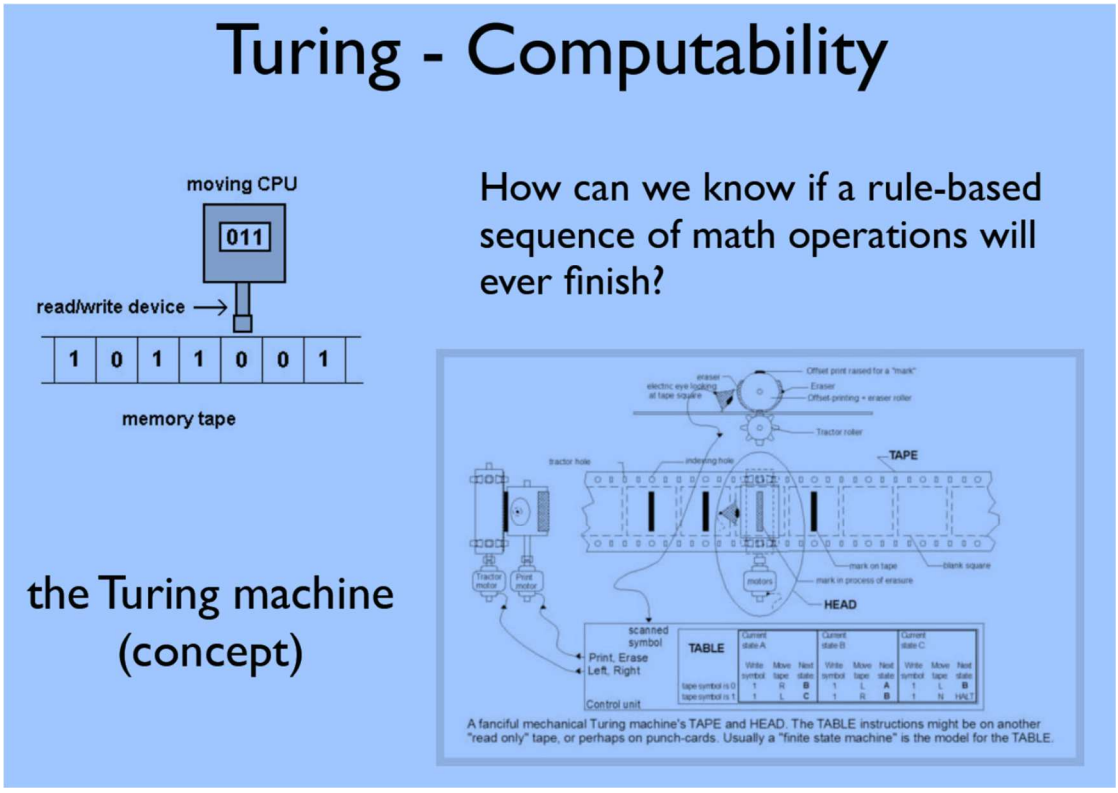
(i) A számítógép legyen teljesen elektronikus.
(ii) Használjon kettes számrendszert.
(iii) Legyen belső adattárolója (memóriája)
(iv) A memória tárolja a feladatokat (a programokat) is.
(v) Legyen a számítógép univerzális Turing-gép.

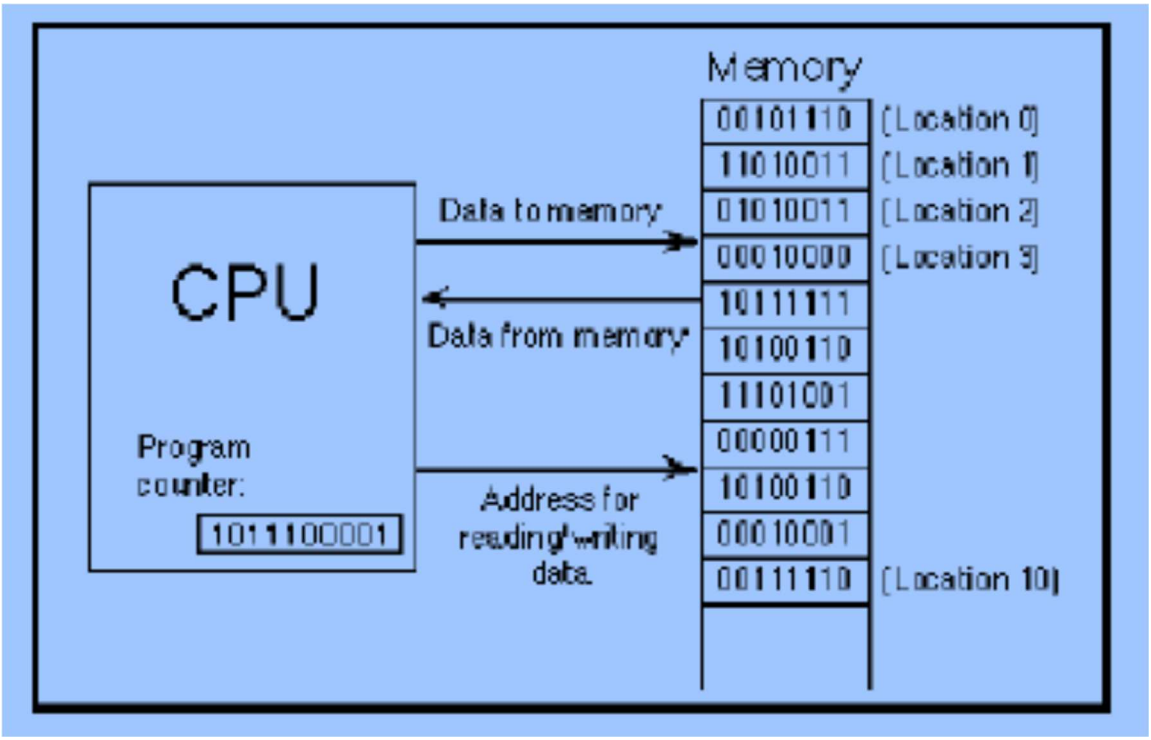
Harvard architektúra, SHARC (Neumann (Princeton) - Harvard -- CISC-RISC) mátrix

Beágyazott rendszerek
a mikrokontroller, mint rendszerelem
beágyazott rendszerek és a cloud
Multi-mikroprocesszoros rendszerek – Amdahl törvénye
multiprocesszoros rendszerek -- Amdahl
Magok
Szálak

Forrás: - haszn ---- Turing Neumann Machine ábra Computers_Intro_011.pdf





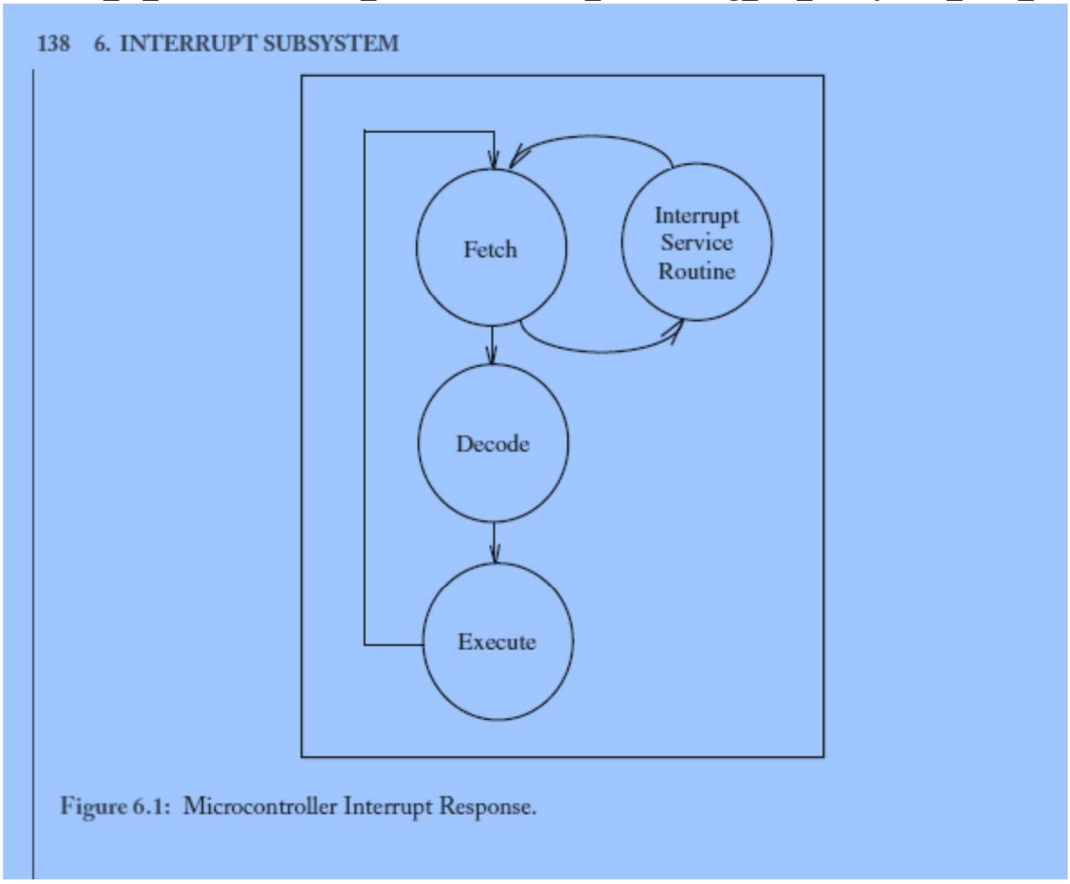

Memory
00101110
11010011
01010011
00010000
10111111
10100110
11101001
00000111
10100110
00010001
00111110
(Location 0)
(Location 1)
(Location 2)
(Location 3)
(Location 10)
CPU
Program counter:
1011100001
Data to memory
Data from memory
Address for reading/writing data

Mikroprocesszoros digitális rendszer rendszerek elemei

Mikroprocesszor, memóriák (RAM, ROM), I/O eszközök, kiegészítők (órajelgenerátor, latch-ek, kétirányú tri-state erősítők, dekódoló áramkörök, stb.)

Forrás:

haszn Steven_F._BarrettArduino_Microcontroller_Processing_for_Everyone!_Part_II



Forrás: - haszn -- sparc architecture ## instruction cycle .ppt

Instruction Execution

- Fetch-Decode-Execute cycles:
  - Fetch the next instruction from memory.
  - Change PC to point to next instruction.
  - Determine the type of the instruction fetched.
  - Find where the data being used by the instruction is kept.
  - Fetch the data, if required.
  - Execute the instruction.
  - Store the results in the appropriate place.
  - Go to step 1 and start all over again.

Instruction Cycles

```
pc = 0;
do {
      ir := memory[pc];        { Fetch the instruction. }
      pc := pc + INSTR_SIZE;   { Move PC to next instruction. }
      decode(ir);              { Decode the instruction. }
      fetch(operands);         { Fetch the operands. }
      execute;                 { Execute the instruction. }
      store(results);          { store the results. }
} while(ir != HALT);
```

Sparc Architecture Overview

- Load/Store architecture
  - ALU cannot access data from memory directly.
  - Data must be loaded into registers before computing.
- RISC (Reduced Instruction Set Computer) architecture
- All instructions are one word (32 bits).
- 5-stage Pipelining CPU.

Forrás: - haszn ------ State diagram of excution SEQUENCING -- FETCH---EXEC -- ezt kerestem ISA .pdf, University of Manchaster

Sequencing

The exact sequencing of operations depends on the microarchitecture.

- The following examples are used to *illustrate some possible* state sequences without trying to optimise too much.
- The original microarchitecture took significantly more cycles than are indicated here.
  - Technology limits meant numerous cycles needed for 'internal' operations.
  - Cheap transistors/wiring help with speed-up

Sequencing is done by a Finite State Machine which uses the IR to determine its path.

The FSM can become quite complex

- sometimes complexity may be traded for performance
- more often the other way around though!

Sequencing

In the following description some more registers are used which are non-architectural (i.e. not 'visible' to a programmer).

- IR the instruction register - holds (the first byte of) the op-code.
- W, Z working registers which may act as a 16-bit pair.

```
0000 78          ld  a, b          ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; ALU:     A  := B;

0001 04          inc b             ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; ALU:     B  := B + 1;

0002 C6 12       add a, 12         ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; OP1:     Z  := [PC]; PC := PC + 1;
                                   ; ALU:     A  := A + Z;

0004 32 00 10    ld (1000), a      ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; OP1:     Z  := [PC]; PC := PC + 1;
                                   ; OP2:     W  := [PC]; PC := PC + 1;
                                   ; MEMO:    [WZ] := A;

0006 BB          cp  a, e          ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; ALU:     flags set by (A - E);

0007 CA 68 24    jp  z, 2468       ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; OP1:     Z  := [PC]; PC := PC + 1;
                                   ; OP2:     W  := [PC]; PC := PC + 1;
                                   ; JUMP:    IF (Zflag == 1) PC := WZ;

000A 96          sub a, (hl)       ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; MEMO:    Z  := [HL];
                                   ; ALU:     A  := A - Z;

000B C5          push bc           ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; ALU:     SP := SP - 1;
                                   ; MEMO:    [SP] := B;
                                   ; ALU2:    SP := SP - 1;
                                   ; MEM2:    [SP] := C;

000C CD 57 13    call 1357         ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; OP1:     Z  := [PC]; PC := PC + 1;
                                   ; OP2:     W  := [PC]; PC := PC + 1;
                                   ; ALU:     SP := SP - 1;
                                   ; MEM1:    [SP] := PC(high byte);
                                   ; ALU2:    SP := SP - 1;
                                   ; MEM2:    [SP] := PC(low byte);
                                   ; JUMP:    PC := WZ;

000F C9          ret               ; FETCH:   IR := [PC]; PC := PC + 1;
                                   ; MEMO:    Z := [SP];
                                   ; ALU:     SP := SP + 1;
                                   ; MEM1:    W := [SP];
                                   ; ALU2:    SP := SP + 1;
                                   ; JUMP:    PC := WZ;
```

State machine code

The Verilog code to implement the machine here (and following) is not specified exactly here. A next-state *decision* is made in the `fetch` state, before the IR is loaded.

To make this 'clean' code would involve adding an extra `decode` state, after the fetch.

```
always @ (posedge clk)
  case (state)
    `FETCH:  begin
             IR <= mem_data_in     // Could be done here ...
             PC <= PC + 1;         // ... or in separate 'always' block
             state <= `DECODE;
             end

    `DECODE: begin
             case (IR[7:6])
               2'b00: ...
               2'b01: ...
               2'b10: if (IR[2:0] == 3'b110) state <= `MEM0;   // (HL)
                      else                   state <= `EXEC0;  // reg.
               2'b11: ...
             end

    ...
```

You can, no doubt, think of other approaches which may save the 'extra' state. For example:

- Do the first execution step in a '`case`' within the '`decode`' state.

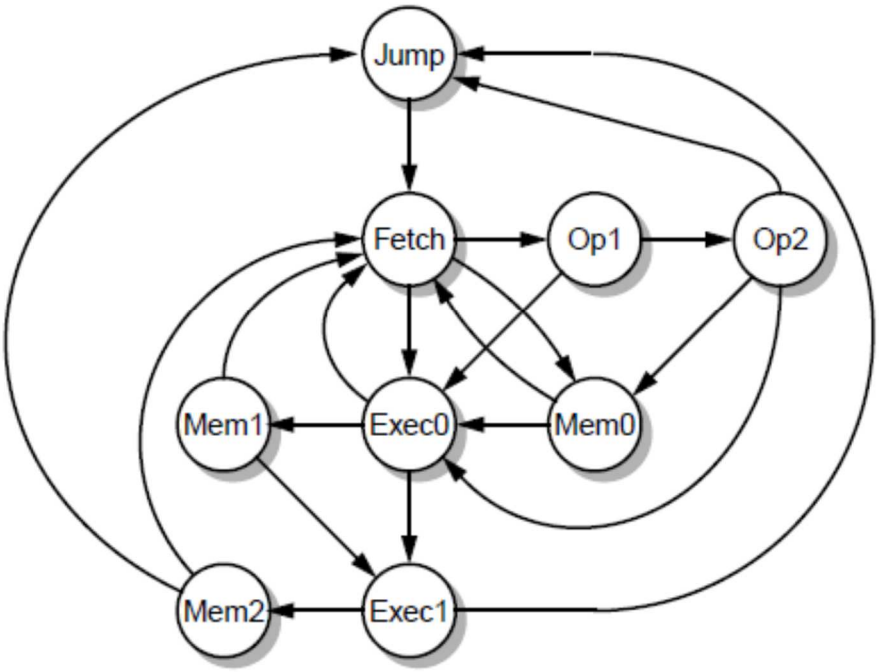
or

- Make a state decision in '`FETCH`' on the IR *before* it is latched i.e. use '`mem_data_in`' in this step only (IR thereafter).

or

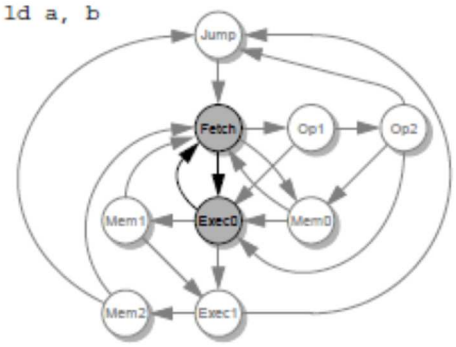
- ...

Possible state diagram

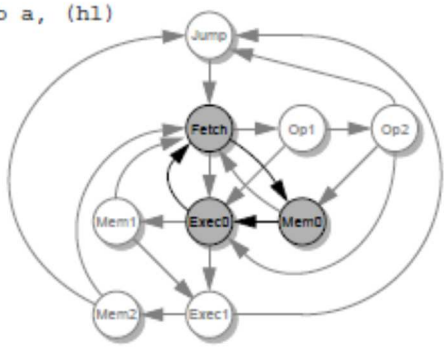


Possible execution paths

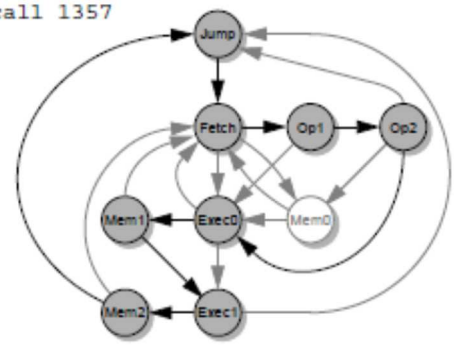
```
ld a, b
```



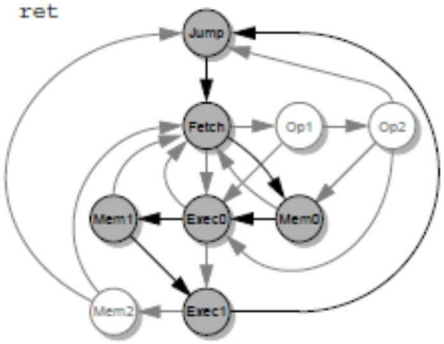
sub a, (hl)



call 1357

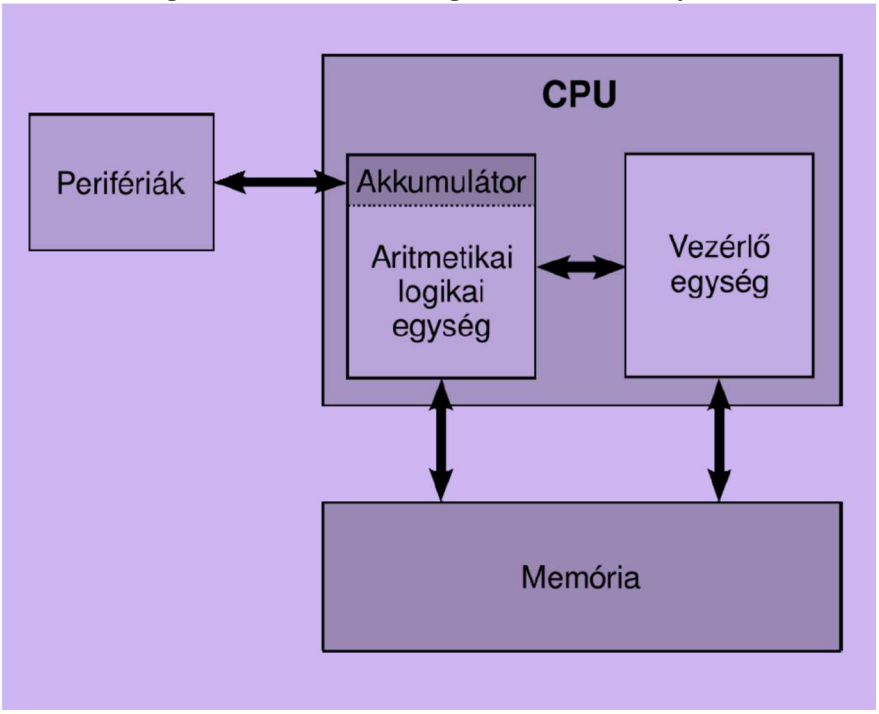


ret



Példák
a 8085-ös mikroprocesszor
a 8051-es mikrokontroller

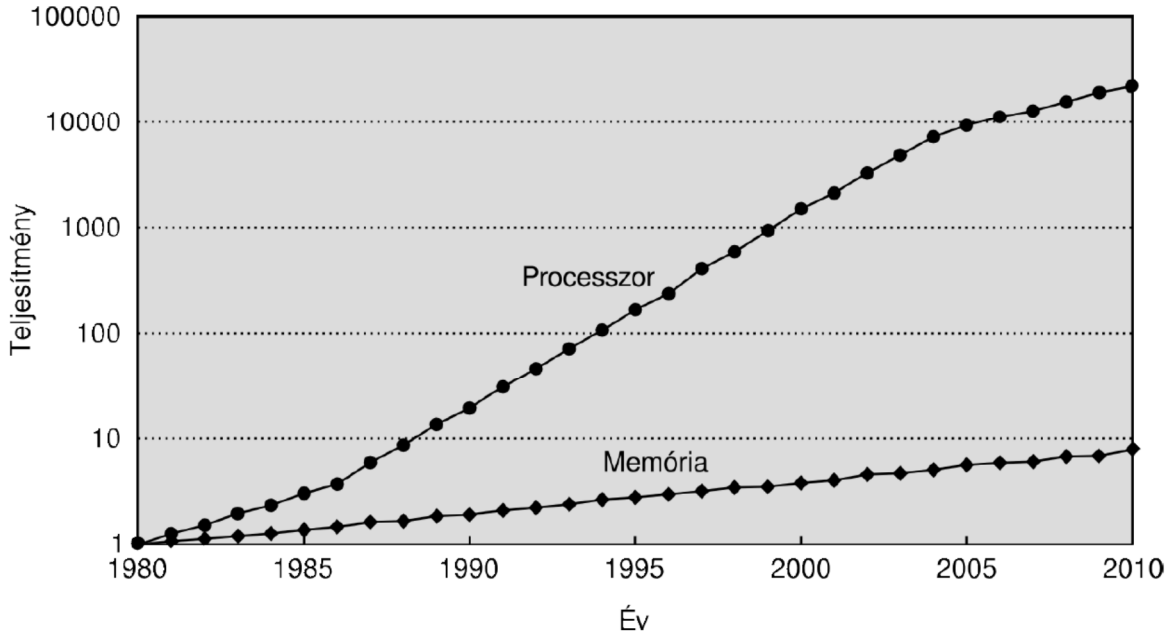
Forrás - haszn ---- BME szga1 Információ feldolgozási modellek .pdf



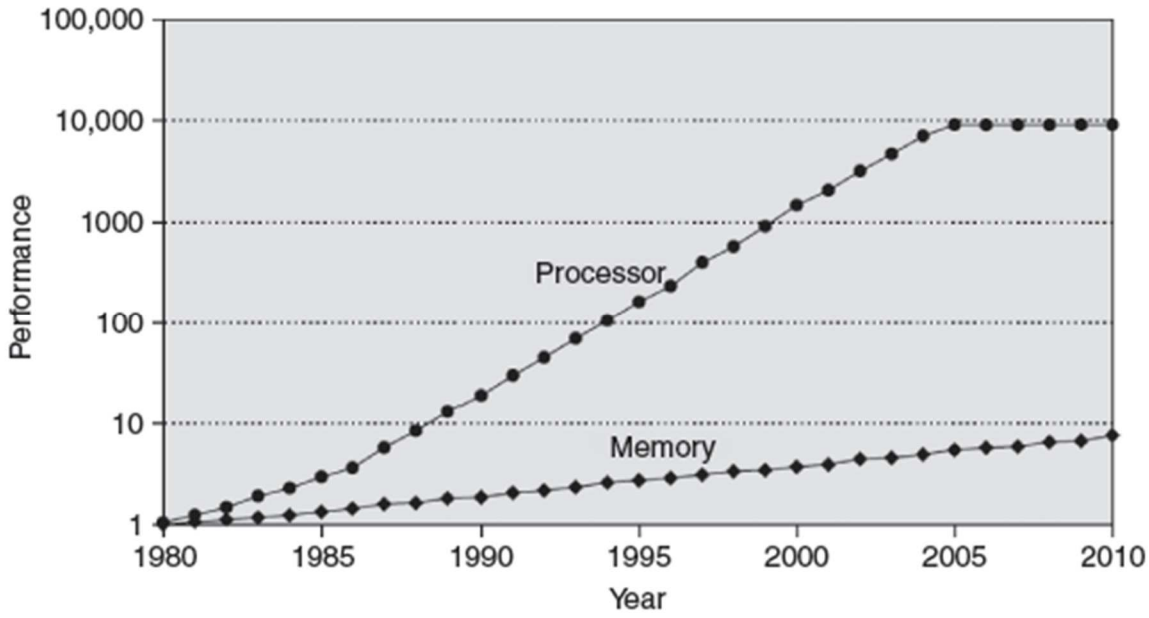
Az utasítások az adatokkal együtt vannak eltárolva a memóriában
Memória tartalma:
- Fix bitszélességű adategységek
    - Lehet
        - Utasítás
        - Egész szám
        - Karakter
        - Lebegőpontos
        - …
- Megkülönböztetés: NINCS!
- Attól függ, milyen utasítás nyúl hozzá

Forrás - haszn ---- BME szga1 Információ feldolgozási modellek .pdf

- Szűk keresztmetszet:
  - **Memória-sávszélesség!**

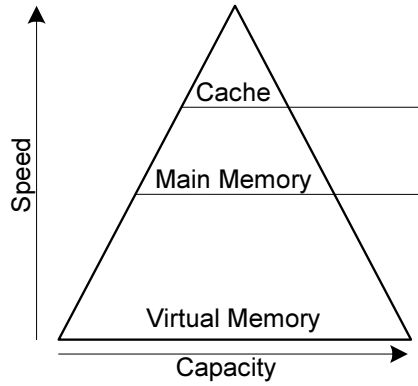


forrás ELSEVIER -- nem ugyanaz --



Moore törvény → Sima Dezső cikke "Core-count Law"

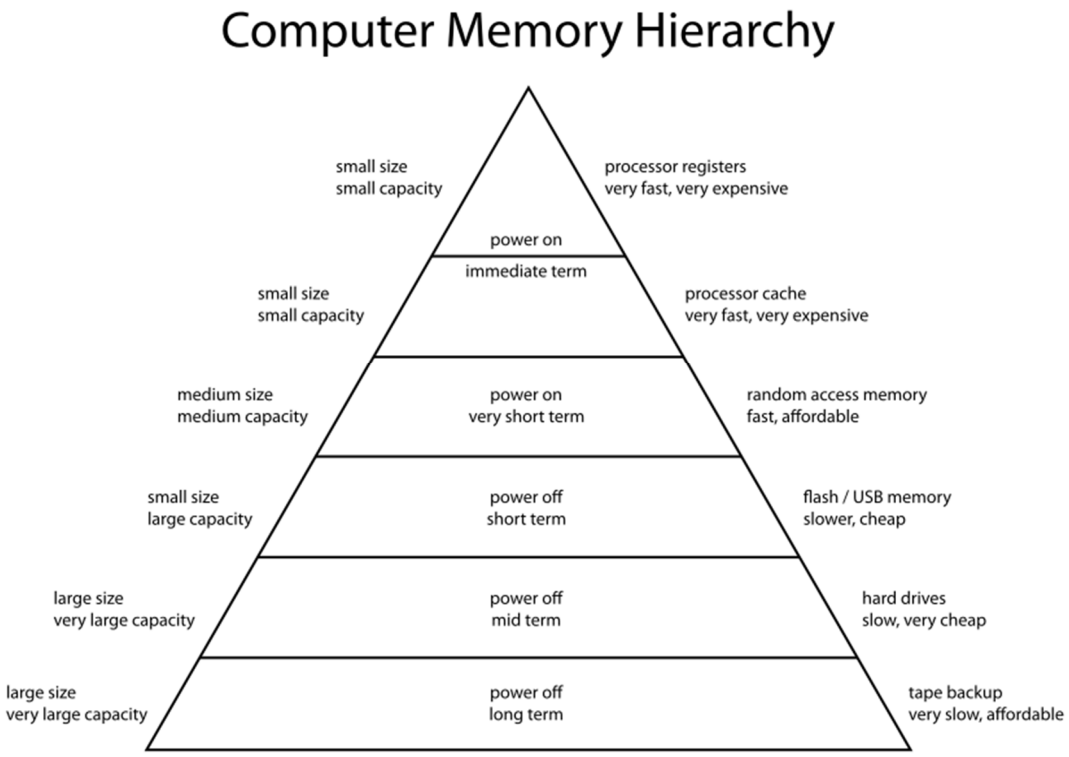
Forrás: Elsevier??

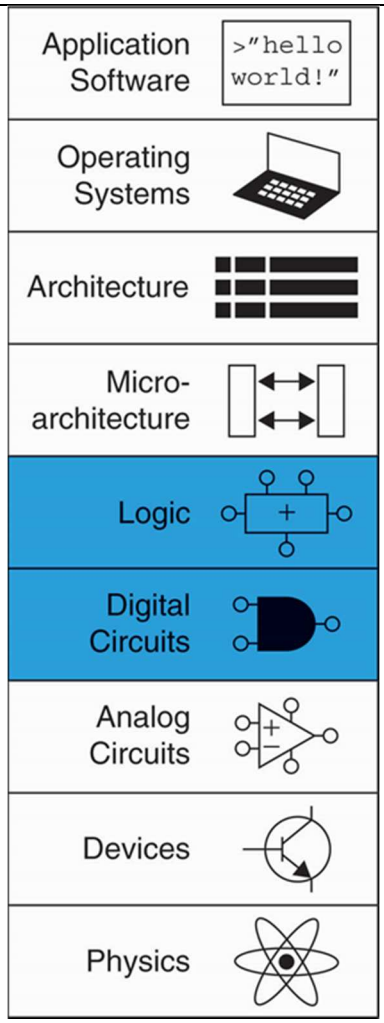
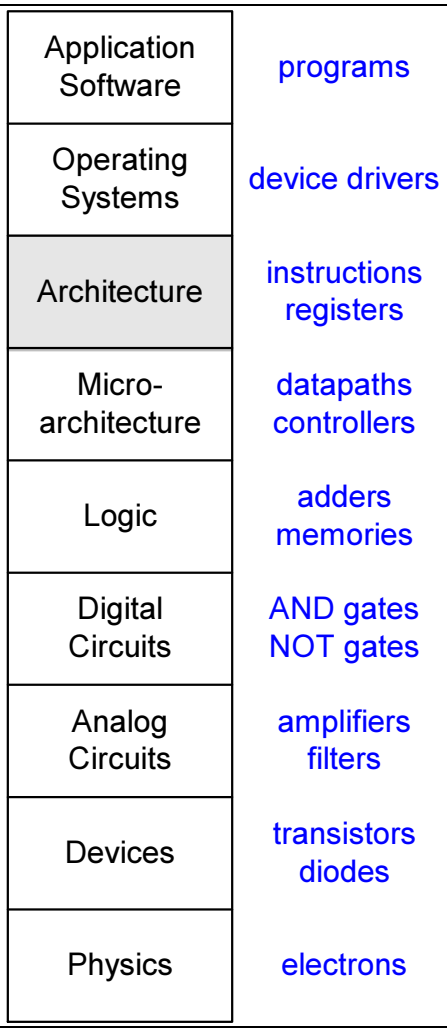


Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10 - 50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

- **Physical Memory:** DRAM (Main Memory)
- **Virtual Memory:** Hard drive
  - Slow, Large, Cheap

Forrás: Wikipedia …

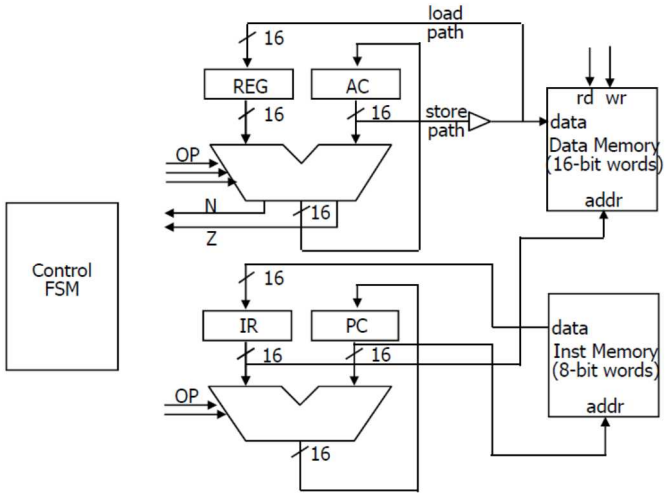


Forrás: Elsevier …	Forrás: Elsevier átalakítva
	

Forrás: - haszn ---- utasitasok vegrehajtasa Neumann Harvard CISC RISC .pdf

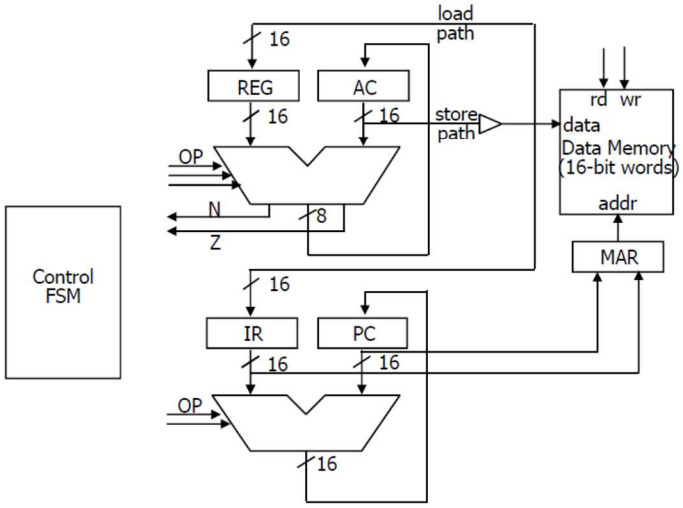
Block diagram of processor (Harvard)

- Register transfer view of Harvard architecture
  - Separate busses for instruction memory and data memory
  - Example: PIC



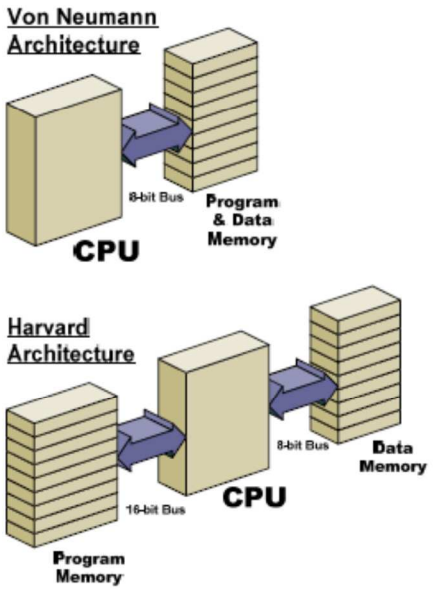
Block diagram of processor (Princeton)

- Register transfer view of Princeton / von Neumann architecture
  - Single unified bus for instructions, data, and I/O
  - Example: MSP430



Forrás: - haszn ---- Harvard PIC intruction cycyles review_tutorial_3_per_page.pdf

Harvard Architecture



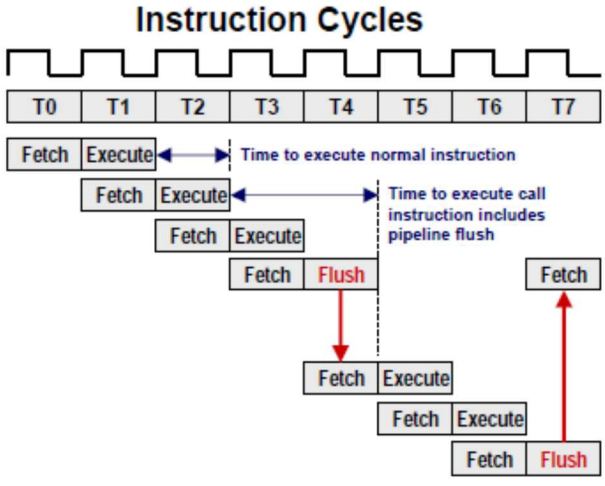
- **Von Neumann Architecture:**
  - Fetches instructions and data from a single memory space
  - Limits operating bandwidth
- **Harvard Architecture:**
  - Uses two separate memory spaces for program instructions and data
  - Improved operating bandwidth
  - Allows for different bus widths

Instruction Pipelining

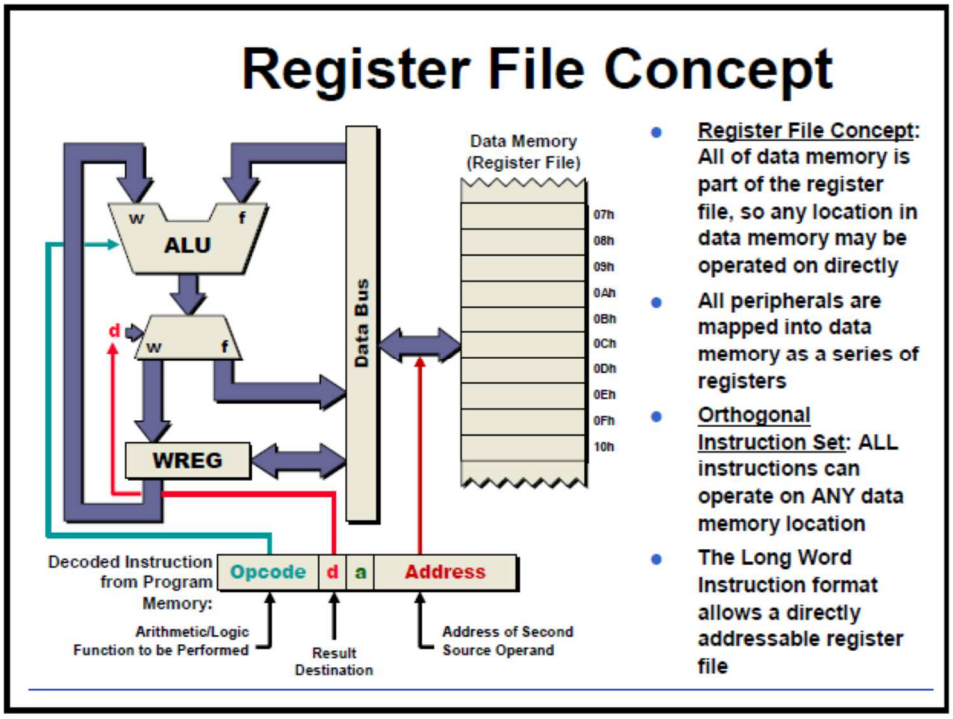
- **Instruction fetch is overlaped with execution of previously fetched instruction**

Example Program

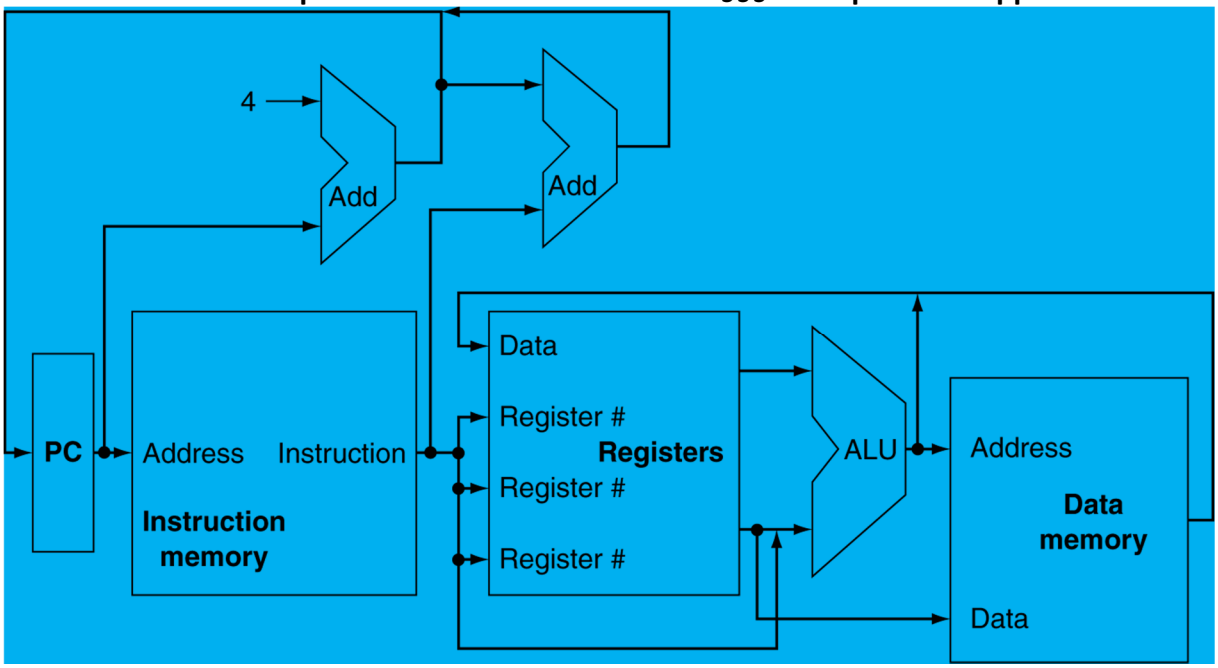
```
1   MAIN  movlw  0x05
2         movwf  REG1
3         rcall  SUB1
4         addwf  REG2

51  SUB1  movf   PORTB,w
52        return
53  SUB2  movf   PORTC,w
54        return
```



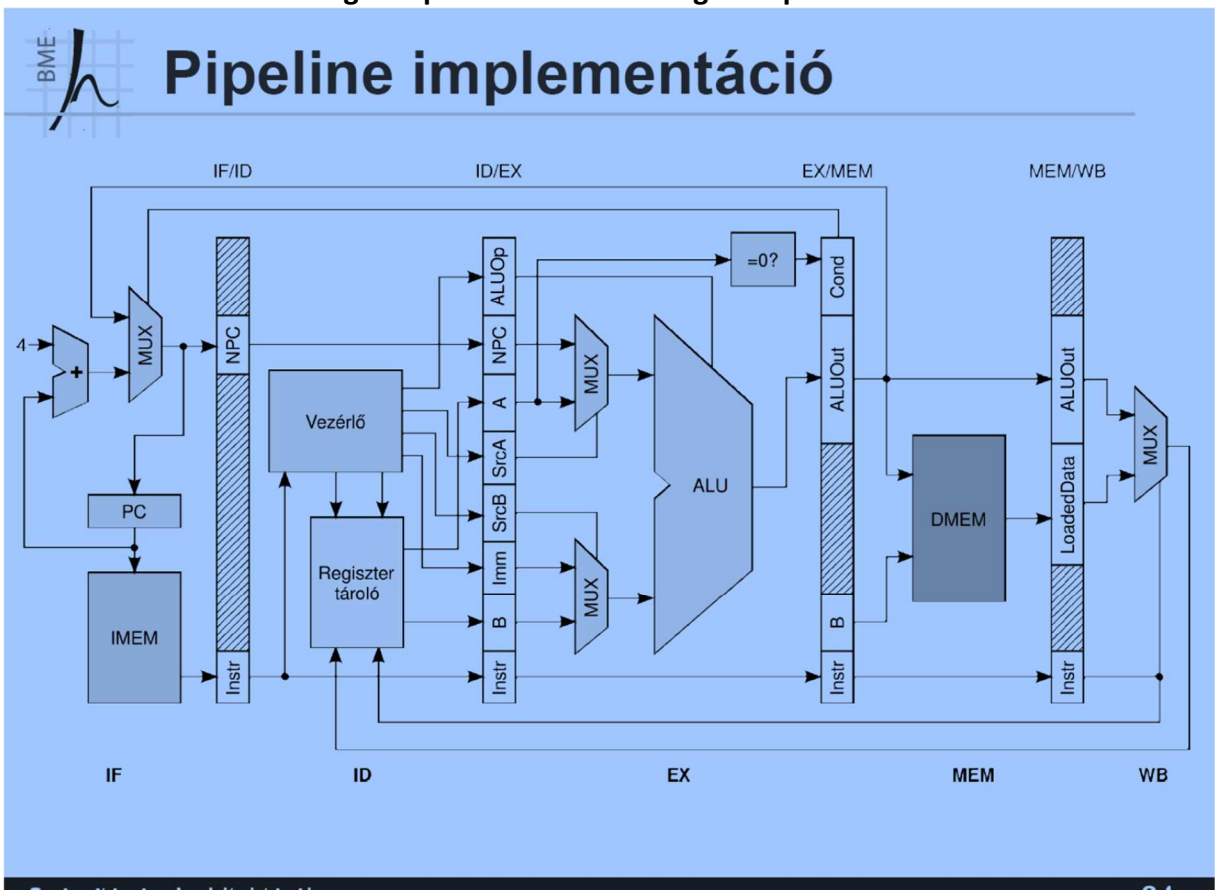
Forrás: - haszn ---- Harvard PIC intruction cycyles review_tutorial_3_per_page.pdf



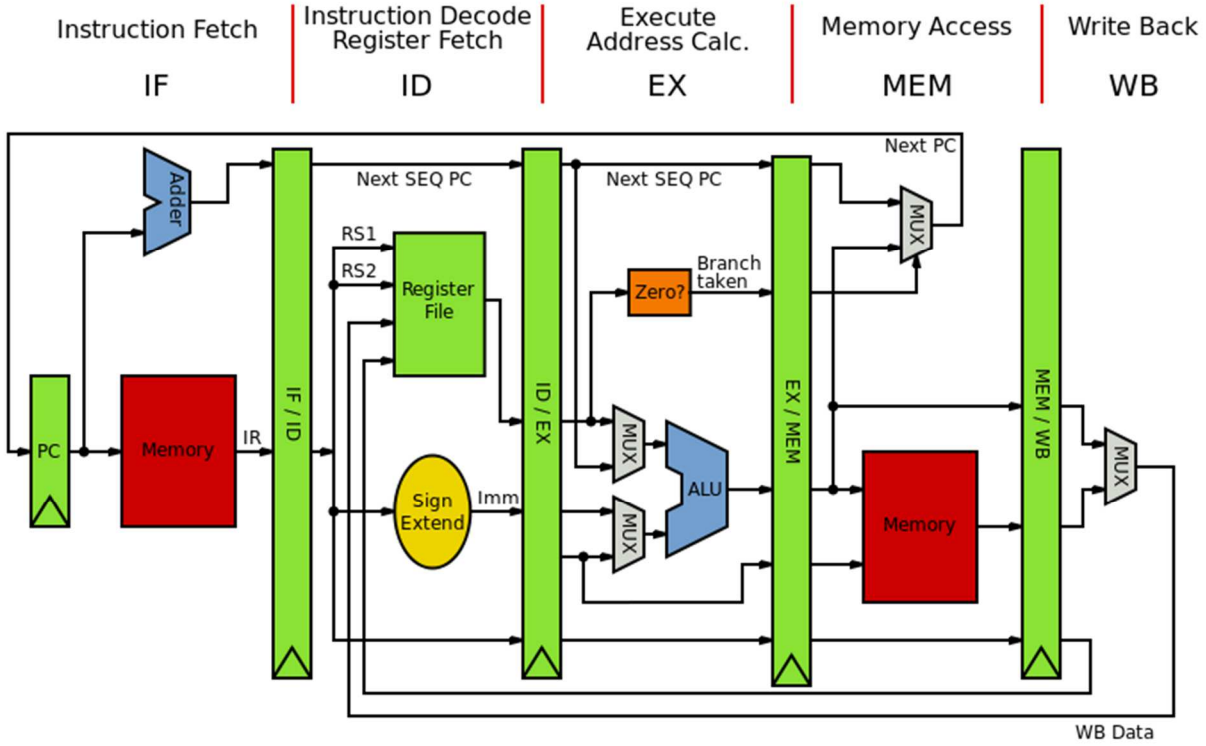
Forrás: - haszn ---- Chapter 4 The Processor HARVARD §§§ MIPS processor .ppt



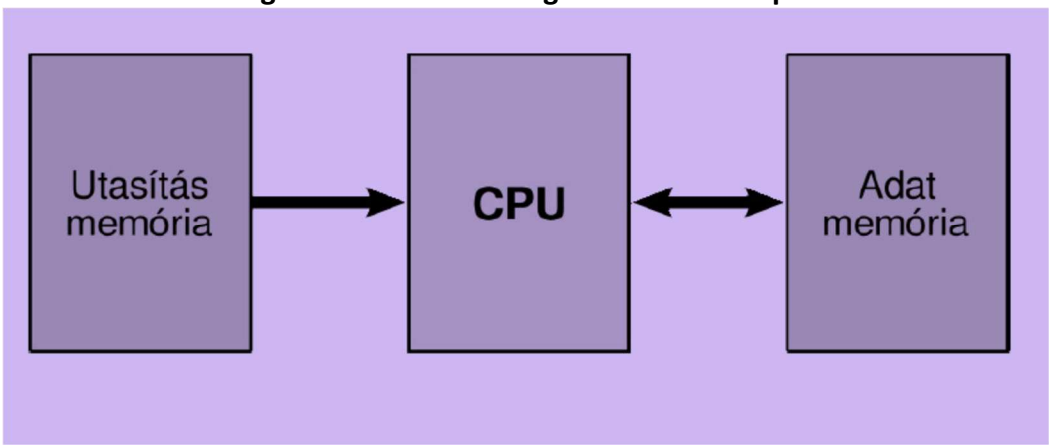
Forrás: - haszn ---- BME szga8 Pipeline utasításfeldolgozás .pdf



Forrás: - haszn ------ NAGYON jóó 1-2-3 BUSES Hardwired Control Microprogramming Automatacontrol CPU_Design.pdf



Forrás: - haszn ---- BME szga1 Információ feldolgozási modellek .pdf



Eltérés a Neumann architektúrához képest:
Az utasítások és az adatok tárolása fizikailag elkülönül egymástól

Klasszikus Harvard architektúra:

- Utasítások: kizárólag az utasításmemóriában
- Adatok: kizárólag az adatmemóriában

A kétféle memóriaművelet átlapolható:

- Amíg lehívja az i. utasítás operandusát
- Azzal egy időben lehívhatja az i+1. utasítást

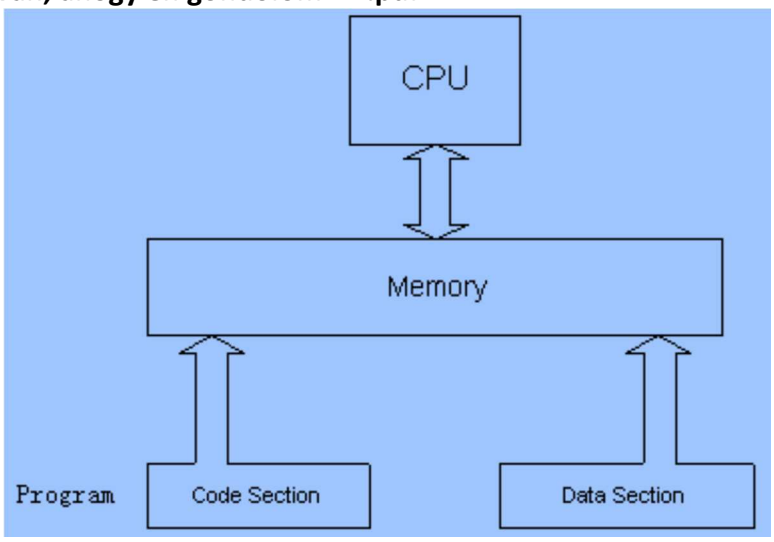
Módosított Harvard architektúra:

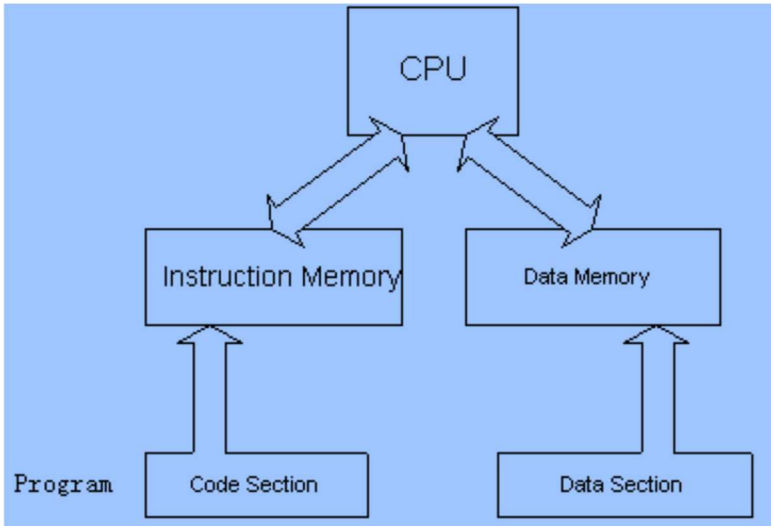
- Utasítás memóriából is lehet adatot olvasni

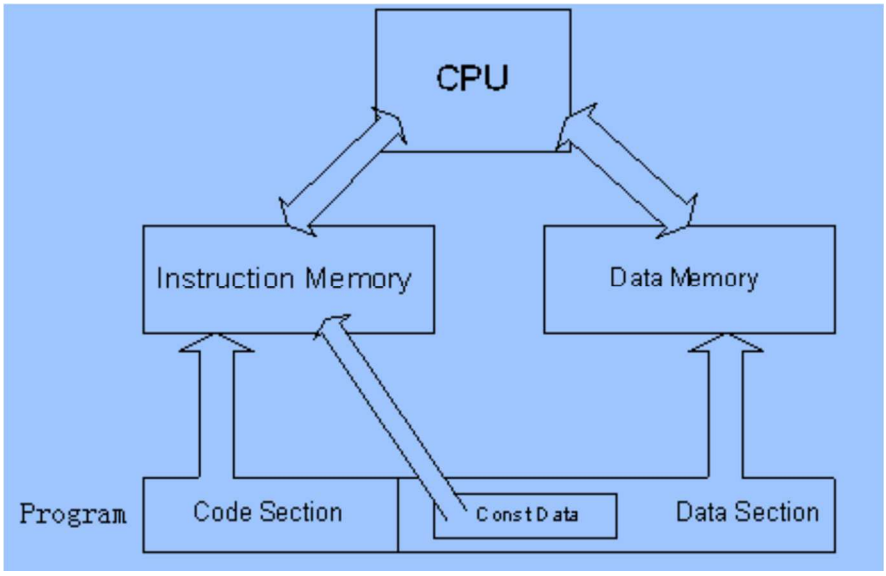
 Alkalmazás:

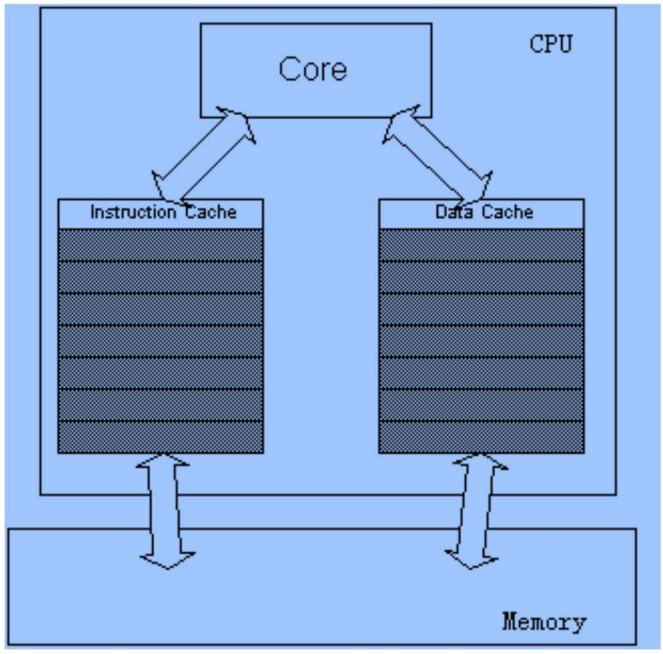
- Mikrokontrollerek, jelfeldolgozó processzorok (PIC, Atmel, ...)

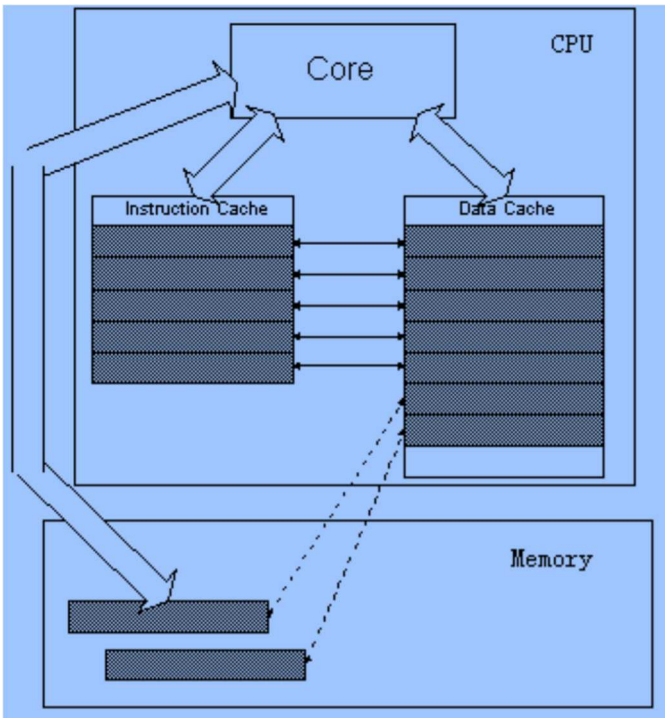
Forrás: -haszn ---- Harvard_Computer_Architecture -rövid TANULMÁNY -- nem tul szivonalas , de EBBEN úgy van, ahogy én gondolom --- .pdf

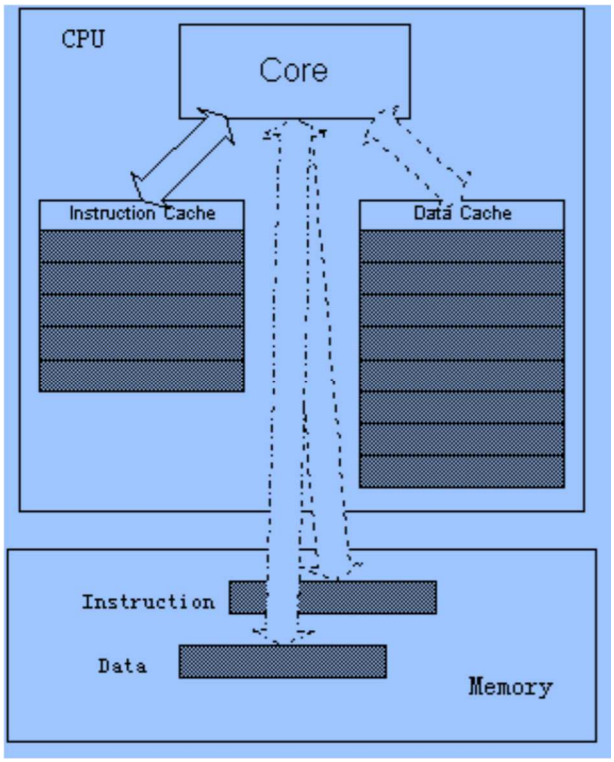







CPU
Core
Instruction Cache
Data Cache
Memory


CPU
Core
Instruction Cache
Data Cache
Memory



8051

Harvard, pipeline, RISC

Forrás : Benesóczky

Az előző fejezetben ismertetett vezérlőkkel a megoldható feladatok köre elég korlátozott. Felmerült az igény olyan univerzálisan használható vezérlőre, amely nagy tömegben és olcsón gyártható. Erre a célra fejlesztették ki a speciális adatstruktúrával kiegészített vezérlőket, a mikroprocesszorokat. A mikroprocesszor elődjének a korai számítógépek központi egysége (Central Processing Unit) tekinthető. Amikor a technológia képes lett, a számítógép központi egységének egyszerűsített architektúráját egyetlen chipen megvalósítani, akkor jött létre az első mikroprocesszor (μP, CPU). Manapság a nagyszámítógépek központi egységét is mikroprocesszor(ok) valósítják meg, ezeket a rendkívül bonyolult áramköröket szuper mikroprocesszoroknak nevezik. Itt csak az egyszerűbb, főként vezérlési feladatok megoldására alkalmazható CPU-kkal foglalkozunk.

A mikrokontrollerek (μC-k) olyan áramkörök, amelyeknél egy IC-ben megtalálható a mikroprocesszor, egy kisméretű RAM (kb. 64 byte - 2 kbyte), többségüknél ROM EPROM vagy EEPROM (kb. 2-16 kbyte) és több olyan periféria, amelyek segítségével egyszerűbb vezérlések alakíthatók ki, viszonylag kevés egyéb áramköri elem felhasználásával.

A mikrokontrollereket a mindennapi környezetünkben sok helyen megtalálhatjuk, alkalmazzák az audio és video magnók vezérlésére, kapcsoló órákban, a CD lejátszókban, a winchesterekben stb.

A különféle mikrovezérlőkben leggyakrabban előforduló perifériák:

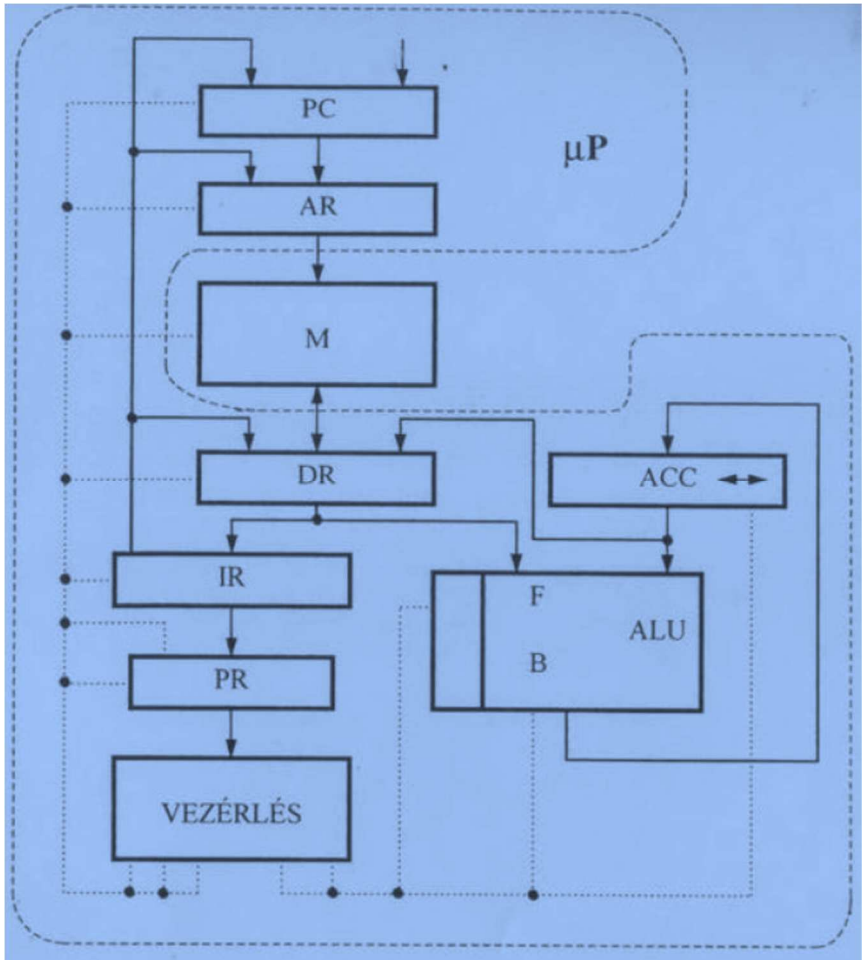
- párhuzamos ki és bemeneti portok,
- időzítő áramkörök (timerek),.
- kétirányú soros vonali interfész (aszinkron és szinkron),
- A/D, D/A konverter,
- LCD meghajtó.

Az utasításkészletben egyrészt a processzoroknál megszokott általános célú utasítások (adat mozgató, ugró, aritmetikai stb.), másrészt a saját perifériák kezelését könnyítő speciális célú utasítások találhatók meg.

A továbbiakban az ipari standardnak számító MCS-51 család, néhány tagjával foglalkozunk röviden.

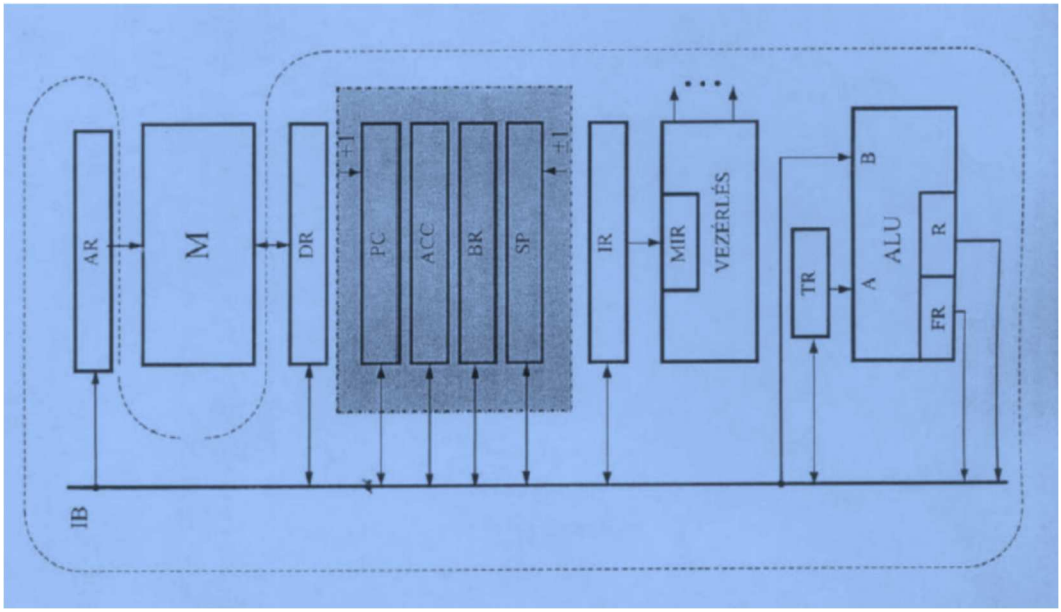
A mikroprocesszor származtatása

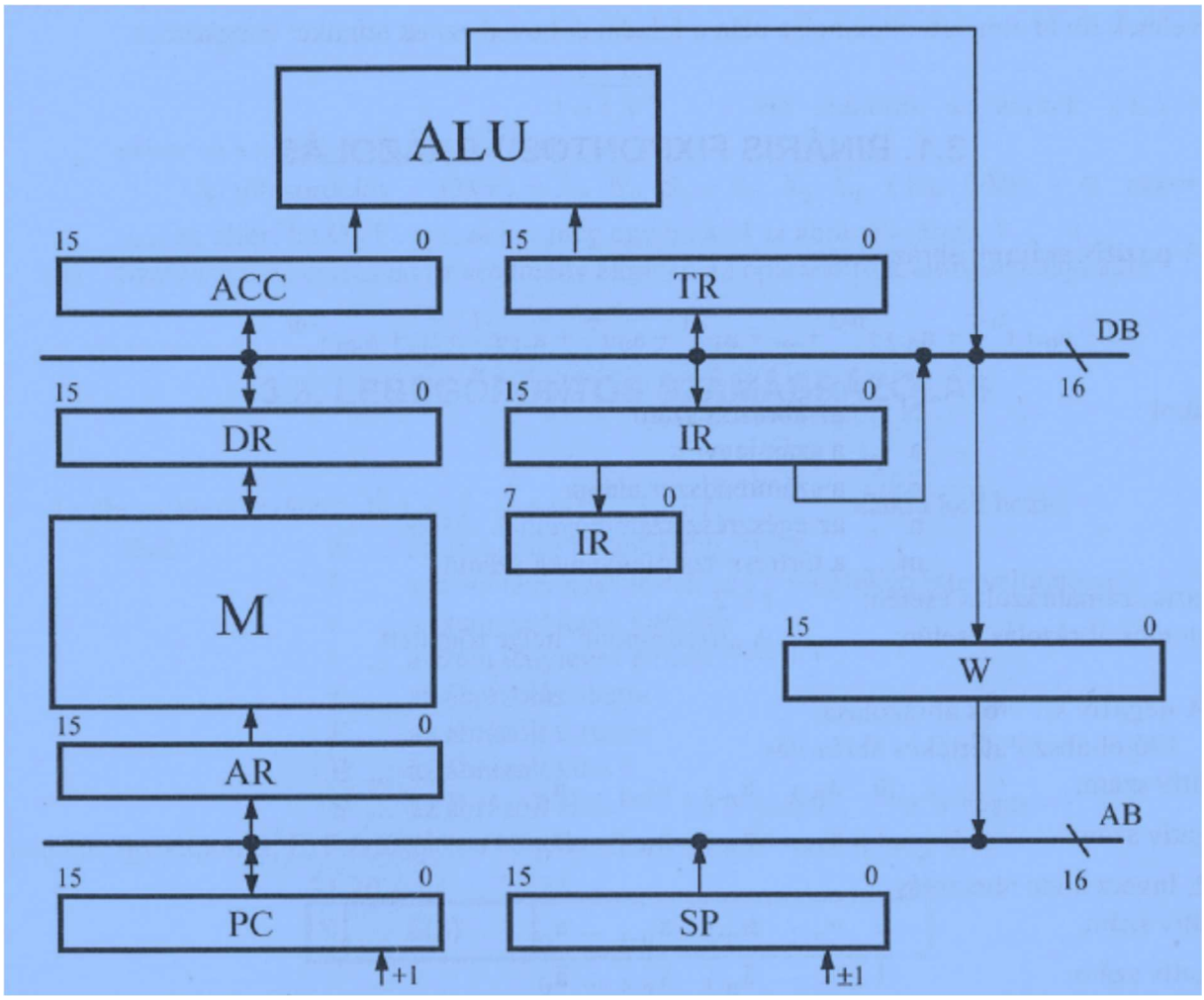
Forrás Horváth L

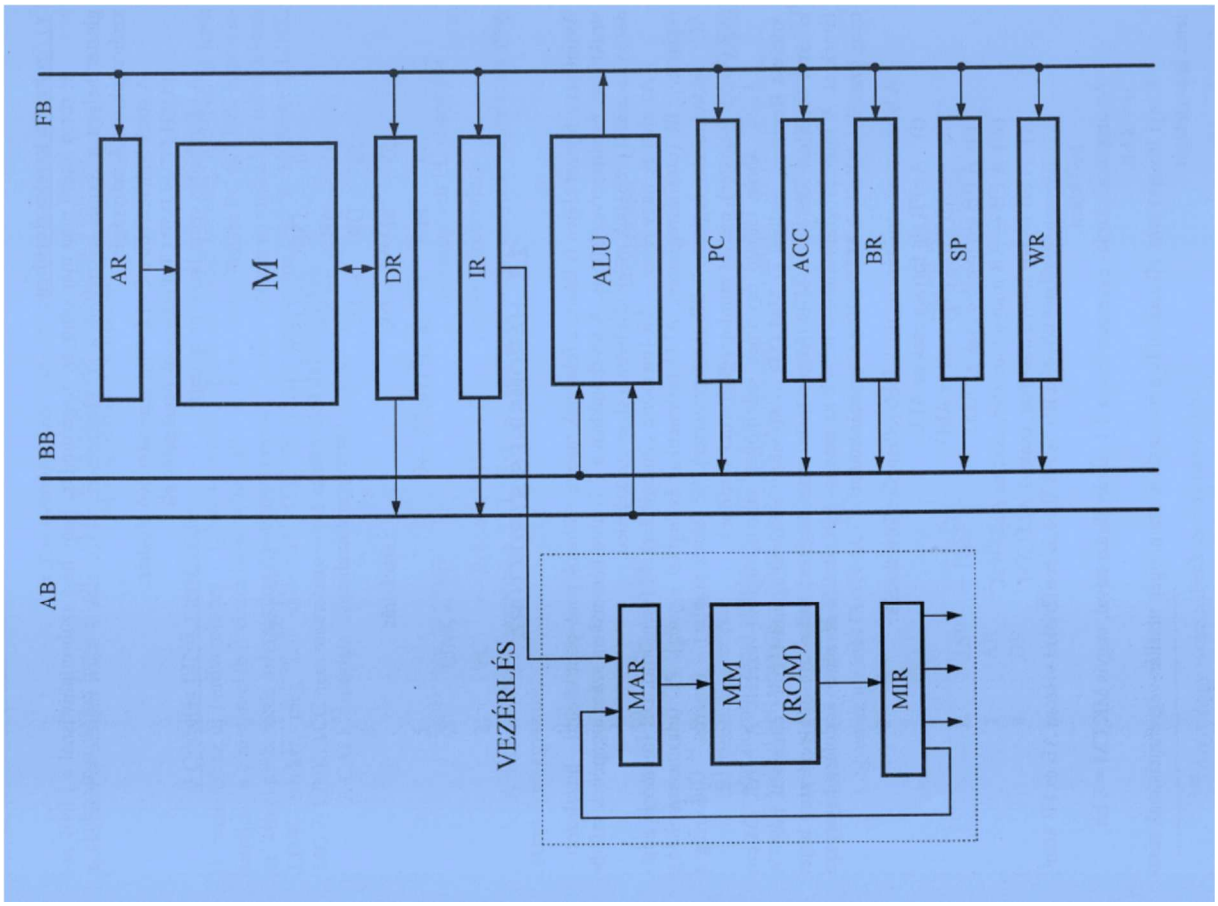


ez itt huzalozott számítógép

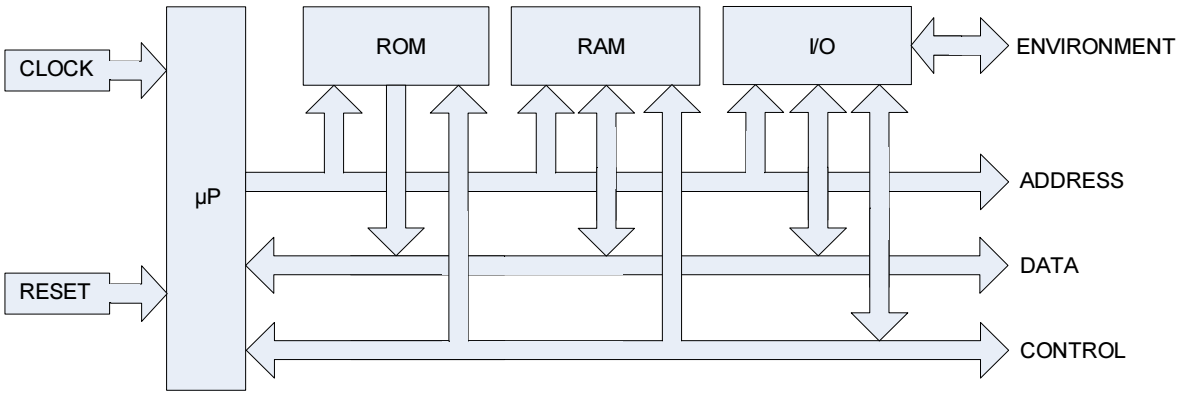
- huzalozott inkább RISC?,
- mikroprogramozott inkább CISC ??




ALU
15
0
ACC
15
0
TR
DB
16
15
0
DR
15
0
IR
7
0
IR
M
15
0
W
15
0
AR
AB
16
15
0
PC
15
0
SP
+1
±1

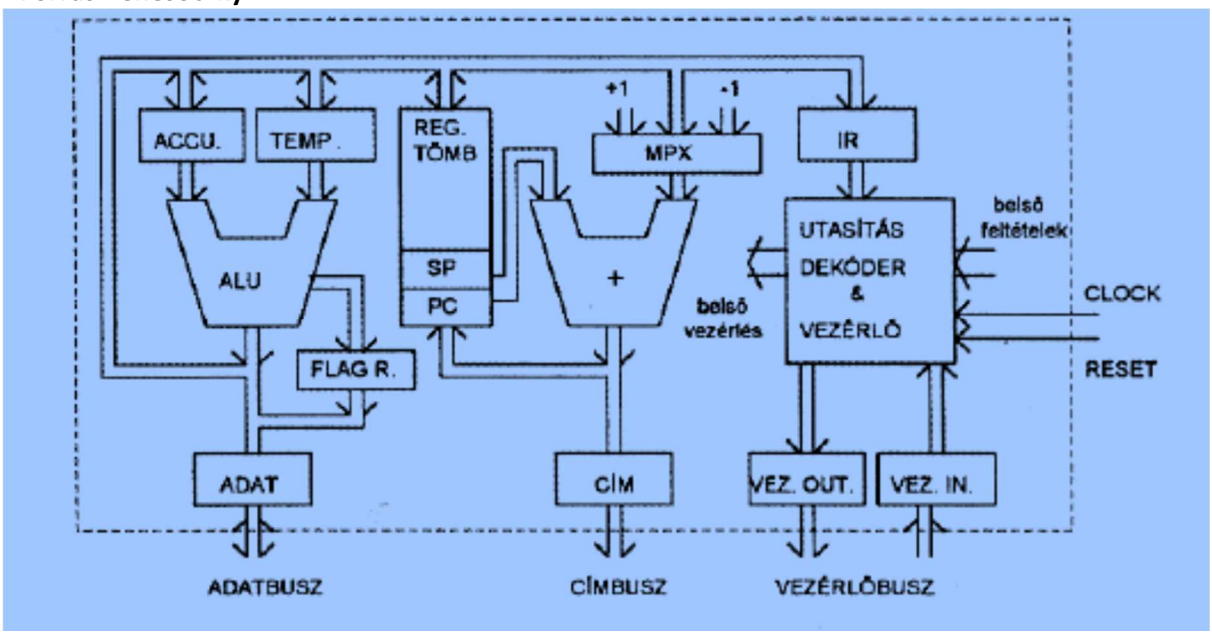


Egyszerű mikroszámítógép blokkvázlata



Mikroprocesszor felépítése

Forrás Benesóczky



Forrás: Manchester University

Specialised processing architectures

- To date you have experienced the principle of processor operation …
  - Fetch, Decode, Execute
  - Integer arithmetic, logical operations {ADD, SUB, MUL, AND, OR …}
- … and seen some of the ways of implementing these

These operations can provide a 'General Purpose' processor

Specific applications demand particular other of processing: for example:

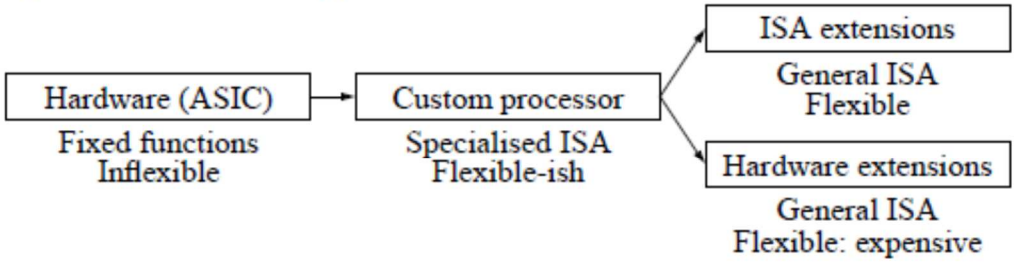
- Numeric processing – floating point arithmetic; vector processing
- Cryptography – 'strange' bit manipulation/transpositions; long word lengths
- Digital signal processing – real-time data streams; fast multiplication
- 'Flashy' graphics – very processor intensive but highly parallelisable
- …

… ez maga a: DIGITAL SYSTEM DESIGN

Forrás: Manchester University

History

Specialised processors appear where there is sufficient application demand and the needs cannot be filled by 'general purpose' devices. The evolution of the implementation of such applications tends to be:



- When something is feasible (and economic) a hardware solution may be built.
- As Moore's Law provides more, faster resources more programmability is introduced
- As general purpose processors become more capable they can assimilate these functions
  - With the addition of extra instructions
  - With additional hardware
    - manufacturer's extension
    - coprocessor – possibly designed by ASIC house

Example: DSPs

Specialised DSPs emerged around 1980 (cf. general purpose microprocessors from early 1970s). Early unit – a classic example was the TMS32010 – were very poor at 'general' code but provided high performance over a range of DSP applications. Early units were integer/fixed point; later floating point processing was incorporated. Later examples became closer to 'general' microprocessors whilst retaining DSP features (see later). 'Dedicated' DSPs still exist and are capable of more MIPS than general purpose cores – for a limited range of applications.

The 'explosion' of mobile (radio connected) computers probably helped sustain DSP development.

Often DSPs are used in conjunction with a 'host' processor in a heterogeneous multiprocessor system.

Programming

Programming is 'specialised' architectures is less amenable to 'traditional' languages than that for 'general purpose' microprocessors. This can be due to irregular instruction sets (such as many DSPs) or high degrees of parallelism (e.g. GPUs), often without explicit thread synchronisation. Similar issues can apply to the specialised 'extensions' to general purpose ISAs.

Programming these units often comes down to the use of hand-written (and optimised) libraries which provide an Application Programming Interface (API) to integrate them with 'conventional' code.

Examples, from the graphics world, could be OpenGL or DirectX.

Forrás: Manchaster Universty

Coprocessors

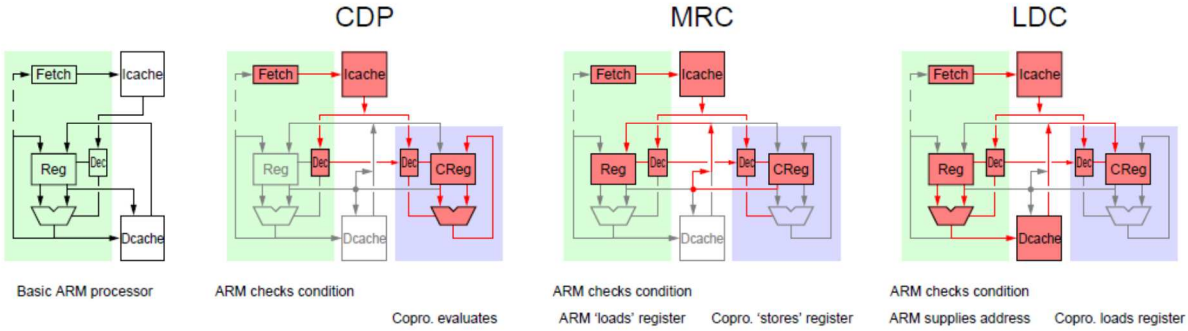
A coprocessor is a processing unit that is not (or, historically, was not) part of the basic ISA

It has (they have) some reserved space in the instruction set, used as needed.

A typical example would be floating point processing

Example: ARM has three classes of coprocessor operations

- Coprocessor data processing (CDP)
- Coprocessor register transfer (MRC/MCR)
- Coprocessor load/store (LDC/STC)



Operations *possible* using pre-existing bus structures.

Forrás: haszn -- DSP Harvard VLIW slides evolution 1997.pdf

Instruction Set

DSP Processor

- Specialized, complex instructions
- Multiple operations per instruction

```
mac x0,y0,a   x:(r0)+,x0   y:(r4)+,y0
```

General-Purpose Processor

- General-purpose instructions
- Typically only one operation per instruction

```
mov *r0,x0
mov *r1,y0
mpy x0,y0,a
add a,b
mov y0,*r2
inc r0
inc r1
```

Addressing

DSP Processor

- Dedicated address generation units
- Specialized addressing modes; e.g.:
  - Autoincrement
  - Modulo (circular)
  - Bit-reversed (for FFT)
- Good immediate data support

General-Purpose Processor

- Often, no separate address generation unit
- General-purpose addressing modes

Forrás: haszn -- DSP Harvard VLIW slides evolution 1997.pdf

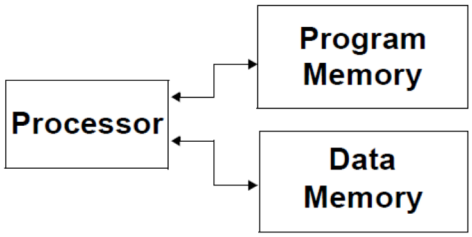
Memory Architecture

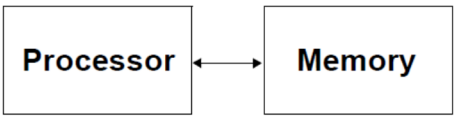
DSP Processor

- Harvard architecture
- 2-4 memory accesses/ cycle
- No caches—on-chip SRAM

General-Purpose Processor

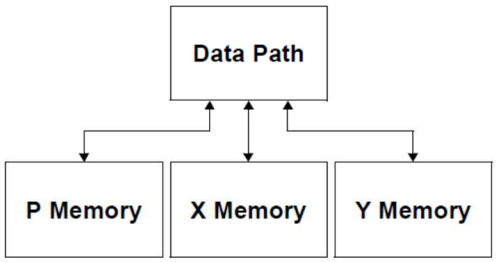
- Von Neumann architecture
- Typically 1 access/ cycle
- May use caches





Second Generation DSPs (1987) Example: Motorola DSP56001

- 24-bit data, instructions
- 3 memory spaces (X, Y, P)
- Parallel moves
- Single- and multi-instruction hardware loops
- Modulo addressing
- 75 ns MAC (21 ns today)

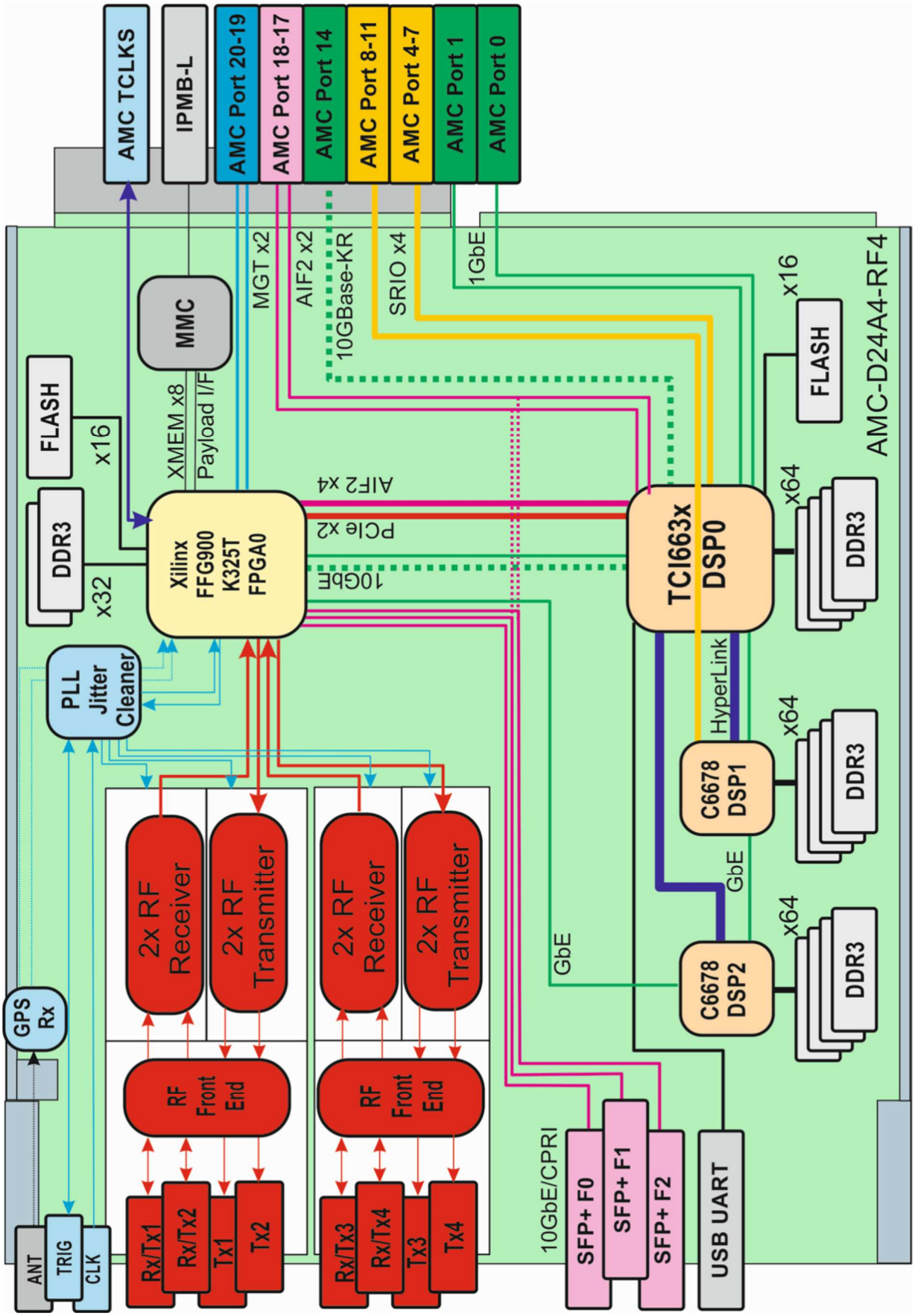



```
move   #Xaddr,r0
move   #Haddr,r4
rep    #Ntaps
mac    x0,y0,a   x:(r0)+,x0   y:(r4)+,y0
```

- Other second-generation processors: AT&T DSP16A, Analog Devices ADSP-2100, Texas Instruments TMS320C50

Forrás: http://www.commagility.com/downloads/CA-AMC-D24A4-RF4.pdf

AMC-D24A4-RF4 ARM, FPGA Xilinx és DSP alapú nagy teljesítményű feldolgozó kártya



CommAgility Ltd
sales@commagility.com
www.commagility.com
Tel: +44 1509 228866

The CommAgility **AMC-D24A4-RF4** is an extremely high performance ARM, DSP and FPGA based processing card which includes four integrated, flexible, wideband RF transceiver channels, all in the compact double width Advanced Mezzanine Card form factor. The module is aimed at LTE, LTE Advanced and 5G systems that require MIMO technologies, and enables complete RF to Layer 3 wireless basestation functionality to be implemented on a single card. The performance and flexibility is ideal for use in test equipment, research and development, or a specialized application such as wireless surveillance.
The module's main processor is the TMS320TCI6636, part of TI's new KeyStone II generation of DSP/ARM SoCs. It includes eight C66x+ DSP cores, as well as four ARM Cortex™-A15 cores for higher layer processing. Two additional TMS320C6678 octal C66x core DSPs maximize the AMC's processing capability, with all processors closely coupled through TI's Hyperlink interface and the Ethernet infrastructure of the card with Serial RapidIO (SRIO) backplane connectivity providing inter-card connectivity. There is also a large Kintex-7 FPGA for additional co-processing and to manage the RF interfaces. Kintex-7 provides an ideal compromise between size, speed and cost, and allows fitting of various FPGA sizes from K160T through to K410T.
CommAgility has ported the RF modules and capabilities of its existing AMC-RF2x2 card to the new module, keeping all the software configurable bandwidth (up to 40MHz+) and flexibility (662MHz to 3839MHz). With the new larger board, it can now support 4 RF channels for 4x4 MIMO configurations, or alternatively each pair of channels can be run at different frequencies, allowing carrier aggregation across different bands. The included RF control, management and processing firmware and software provides a high level RF control API while also leaving most of the board available for customer processing.
A comprehensive DSP Board Support Library is provided along with a full SMP Linux implementation for the ARM cores, for easy integration of 3rd party PHY, stacks and customer applications with the card.
Timing and synchronisation for applications is achieved with the built in GPS receiver, or via the front panel and AMC backplane clock I/O.

+ ehhez
http://www.inf.u-szeged.hu/dti/oktatas/fpga_alapu_digitalis_iranyitas

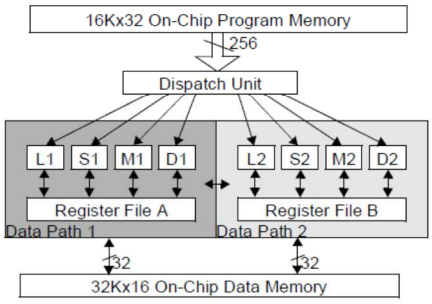
Forrás: haszn -- DSP Harvard VLIW slides evolution 1997.pdf

VLIW

Very long instruction word (VLIW) architectures are garnering increased attention for DSP applications.

Notable recent introductions include Texas Instruments' TMS320C62xx and Philips' TM1000.

Major features:



- Multiple independent operations per cycle
  - Packed into a single large "instruction" or "packet"
- More regular, orthogonal, RISC-like operations
- Large, uniform register sets

The General-Purpose Processor Threat

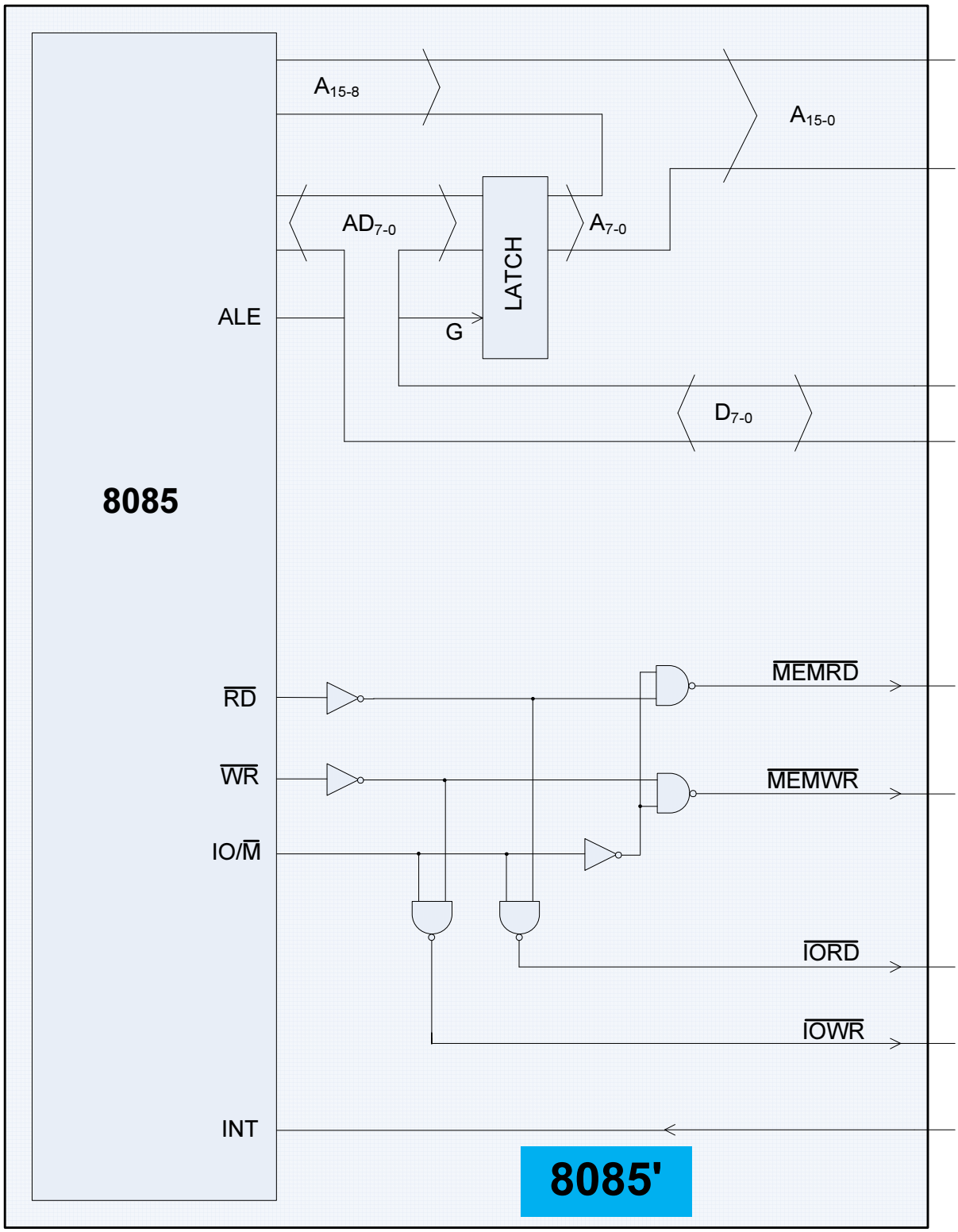
High-performance general-purpose processors for PCs and workstations are increasingly suitable for some DSP applications.

- E.g., Intel MMX Pentium, Motorola/IBM PowerPC 604e

These processors achieve excellent to outstanding floating- and/or fixed-point DSP performance via:

- Very high clock rates (200-500 MHz)
- Superscalar ("multi-issue") architectures
- Single-cycle multiplication and arithmetic ops.
- Good memory bandwidth
- Branch prediction
- In some cases, single-instruction, multiple-data (SIMD) ops.

A 8085’ hipotetikus mikroprocesszor blokkvázlata
HYPO85Z



A 8085' regiszterkiosztás és címzési módok

Regiszterkiosztás -- regiszter modell

A hipotetikus processzornak a következő regiszterei vannak:

A, F, B, C, D, E, H, L regiszterek 8 bitesek, és az SP regiszter 16 bites az alábbi értelmezésekkel:

- A - akkumulátor: kitüntetett regiszter, az aritmetikai és logikai műveletek egyik operandusaként használjuk, és a műveletek eredménye is benne képződik; valamint az IN (perifériáról való bevitel) és OUT (perifériára való kivitel) utasítások implicit operandusa is ez.
- F - jelzőbitek regisztere: a jelzőbitek közül csak a Z-t használjuk, értéke pontosan akkor 1, ha az utolsó aritmetikai vagy logikai művelet eredménye 0. Az A regiszterrel együtt az AF regiszterpárt alkotja, ami verem műveleteknél együtt kezelhető.
- BC, DE, HL - regiszterpárok: együtt 16 bites regiszterként használhatók, de az egyes regiszterek külön-külön is használhatók.
- SP - veremmutató

Címzési módok

- Akkumulátor címzés: az egyik operandus és a művelet eredménye implicit módon az akkumulátor (de nem nevezzük meg). Ezzel találkozunk az aritmetikai és logikai műveleteknél valamint az IN/OUT utasításoknál.

```
AND     B               ; A ← A&B

OUT     38h             ; port38h ← A
```

- Regisztercímzés: operandusként regisztert vagy regiszterpárt adunk meg.

```
LD      A, B            ; A ← B
```

- Közvetlen adatcímzés (immediate): közvetlenül az utasítás kódja után, az utasítás részeként szerepel az operandus

```
LD      B, 10           ; B ← 10
```

- Direkt memóriacímzés: az operandus memóriabeli címét adjuk meg zárójelben (azért kell a zárójelpár, hogy a közvetlen adatcímzéstől meg tudjuk különböztetni)

```
LD      (3000h), D      ; MEM[3000h] ← D
```

- Indirekt címzés: a címet tartalmazó regiszterpár zárójelben megadva szerepel (itt is azért kell a zárójelpár, hogy a regisztercímzéstől meg tudjuk különböztetni).

```
LD      A,(HL)          ; A ← MEM[HL]
```

Mikroprocesszoros rendszerek elemei
RENDSZERTECHNIKAI MEGOLDÁSOK A TELJESÍTŐKÉPESSÉG NÖVELÉSÉRE

A témához ajánlott elolvasni [2] megfelelő fejezeteit. Az óravázlatban hivatkozott ábrák is innen származnak és a **Rendszertechnikai_megoldasok.pdf** fájlban találhatók meg.

CISC és RISC szervezés

(Olvasmány: [2] 2.3. fejezet, 33-36. o.)
Cél: teljesítőképesség növelése
Az utasításkészlet szempontjából vizsgálva, egy program végrehajtási ideje:
P=I□C□T, ahol:
I: a program végrehajtott utasításainak a száma
C: az utasításonkénti órajelciklusok átlagos száma
T: az órajelciklusok hossza
A végrehajtási idő csökkentésére adott technológia mellett két út is van:

CISC szervezés

Egyre összetettebb utasítások alkalmazása. Cél minél több funkciót hardverrel megvalósítani.
Tipikusan mikroprogramozott vezérléssel.
Ekkor I csökken, C nő, T is nőhet (az egyszerű utasításoké is).

RISC szervezés

Kis számú, egyszerű utasítás. Load-store architektúra (műveletvégzés csak regisztereken), huzalozott vezérlés.
Ekkor I nő, de C csökken és T is csökkenhet.
Hardver helyett szoftver! Jól optimalizáló fordítóprogram szükséges. Nagy számú regiszter kell.
Lehetőségek: egyforma hosszúságú (végrehajtási idejű) utasítások: pipeline jó lesz.
Kevesebb hely kell a lapkán, lehetőség van MMU, cache stb. integrálására!
(A korszerű processzorban mindkét megoldás – CISC és RISC – előnyeit ötvözik.)

Szuperskalár architektúra

(Olvasmány: [5]
A szóhossz növekedése miatt egy memóriaszóban több utasítás fér el. Ezek lehívása egy gépi ciklus alatt megtörténik. Ezek egyidejű végrehajtásához a processzor bizonyos funkcionális elemeit (például: fixpontos ALU, lebegőpontos ALU, és ma GPU is) többszörözik. Az utasításlehívás és dekódolás után – amennyiben a utasítások egymástól függetlenek és a szükséges erőforrások rendelkezésre állnak – a *dispatche*r egység egyidejűleg több utasítás végrehajtását is elindíthatja a megfelelő erőforrásokhoz irányítva őket. Ez *MIMD* (Multiple Instruction Multiple Data) működés!
A hardver tervezésekor el kell dönteni, hogy melyik erőforrásból mennyit építsenek be.
A fordítóprogram a program utasításait – amennyiben azok explicite és implicite függetlenek – a processzor architektúrájának ismeretében átrendezheti annak érdekében, hogy az egy utasításszóba kerülő utasítások egyidejűleg végrehajthatók legyenek.
(Az utasítás-egymásrahatások problémájával a pipeline tárgyalásánál foglalkozunk.)

VLIW: Very Long Istruction Word

(Olvasmány: [11])
A CPU-ban szuperskalár architektúrához hasonlóan a processzorban itt is többszörözik a funkcionális egységeket, de azzal ellentétben itt NEM a CPU valamely egységének a feladata, hogy eldöntse, mit lehet párhuzamosan végrehajtani, hanem ezt a feladatot a fordítóprogram végzi el. Az utasításszavak azért hosszúak, mert minden utasításban több funkcionális egységet vezérelnek.

Éppen ezért egyszerűbb hardvert és bonyolultabb fordítóprogramot igényel mint a szuperskalár architektúra.

Társprocesszor

(Olvasmány: [2] 3.2. fejezet, 65-66. o.)
Egy számítógép teljesítőképessége növelhető úgy, hogy az általános célú processzor mellet egy másik, speciális feladatokra (például bonyolultabb aritmetikai műveletek, I/O műveletek) tervezett processzort alkalmazunk: *társprocesszor*. (Például az Intel processzorcsaládban aritmetikai műveletekre: 80287, 80387) Ez NEM multiprocesszoros rendszer, az utasításlehívást és a tárolóhoz való összes hozzáférést (de legalábbis a címszámítást) az általános célú processzor végzi. (Lásd:
3.12. ábra) A társprocesszor haszna az, hogy bizonyos speciális feladatokat gyorsabban képes elvégezni. Általában egyszerre csak az egyik processzor működik, kivétel, ha a társprocesszor működése alatt az általános célú processzor további, kizárólag neki szóló utasításokat hajt végre, de ilyenkor is figyelnie kell a társprocesszor műveletének befejeződésére (például eredmény tárolása). Társprocesszor hiányában a programban a neki szóló műveleteket az általános célú processzor egy megfelelő szoftveres megszakítás kiszolgálásaként a saját utasításkészletével megvalósított algoritmussal (természetesen több lépésben) oldja meg.

Harvard architektúra

(Olvasmány: [2] 5.1. fejezet, 108-109. o.)
Ebben az architektúrában a program utasításait és az adatokat két külön memóriában tároljuk: 5.1. ábra. Ez elvi eltérés a klasszikus Neumann architektúrához képest! (A Neumann architektúra nem tesz különbséget az utasítások és az adatok között, megkülönböztetésük az algoritmusba beépített értelmezés eredménye!) Meggátolja önmódosító programok írását. A teljesítőképességet úgy képes növelni, hogy az i. utasítás végrehajtása során az adat írása vagy olvasása átlapolódik az (i+1). utasítás lehívásával.

SHARC architektúra SCHARC Super Harvard Architectere

Pl

Utasítás pipeline

(Olvasmány: [2] 5.2. fejezet, 110-116. o.)
Egy utasítás feldolgozásának különböző fázisai vannak, például utasításlehívás, dekódolás, végrehajtás. Egy processzor teljesítőképességét növelhetjük, ha egy utasítássorozat egymást követő utasításainak végrehajtási fázisait egymással átlapoljuk: *pipeline*. Ez HIBA lehet!! *SISD* (Single Instruction Single Data) működés. Szemléltető példa: baromfi feldolgozás futószalagon. 5.2. ábra magyarázata szóban.
Megjegyzés: a pipeline MISD -- utasítás-szintű párhuzamosításnál

UTASÍTÁS EGYMÁSRAHATÁS

Az átlapolódó utasítások között előfordulhat *utasítás-egymásrahatás*. Ezeket 3 típusba sorolhatjuk:

Feldolgozási egymásrahatás

Feldolgozási egymásrahatás akkor fordul elő, ha két utasítás végrehajtása ugyanazt azerőforrást igényli. Ilyenkor a sorrendben második utasításnak meg kell várnia az első
végrehajtását. A probléma kezelésére az erőforrásokat (funkcionális egységeket)
többszörözik.

PROCEDURÁLIS EGYMÁSRAHATÁS

*Procedurális egymásrahatás*ról beszélünk, ha egy feltételes ugró utasítás miatt nem tudjuk, hogy a program melyik ágon fog folytatódni. Ilyenkor elvileg megtehető az, hogy mindkét ágról elkezdünk

utasításokat felhozni, de ez meglehetősen erőforrás igényes, ráadásul újabb elágazás után egyre több ággal kellene foglalkozni. A gyakorlatban ilyenkor vagy várunk vagy valamilyen módszerrel megpróbáljuk megjósolni, hogy merre folytatódik a végrehajtás, és arról az ágról folytatjuk az utasítások felhozását, de megjelöljük őket, és ha a jóslás téves volt, akkor eldobjuk az így felhozott utasításokat.

ADAT EGYMÁSRAHATÁS

Adat egymásrahatás tipikus példája, ha egy utasítás eredményét egy rá következő utasítás felhasználja. Ezt észlelni kell és bár a rákövetkező utasítás felhozása és dekódolása megtörténhet, a végrehajtáshoz meg kell várni az előző utasítás végrehajtásának befejezését. Különösen veszélyes, ha egy utasítás átírja (valamelyik) rá következő utasítást, aminek a felhozása már megtörtént. Ezt is észlelni és kezelni kell! (Természetesen az ilyen programozási stílus messziről kerülendő!)

A pipelineban a felhozott utasításokkal együtt visszük az utasításhoz tartozó programszámláló értékét, erre például a relatív ugrások kezelésénél van szükség vagy a fent említett program önmódosítás észlelésénél van szükség.

Vektorprocesszorok

(Olvasmány: [2] 5.3. fejezet, 117-119. o.)

A vektorprocesszorok vektor típusú adatokon dolgoznak, ilyenek elemei között végeznek azonos műveleteket. Míg az utasítás pipeline az egymást követő utasítások végrehajtásának részfázisait lapolja át, a vektorprocesszorok *adatpipeline*t használnak: itt az egymást követő vektorelemek kezelése lapolódik át. A vektorprocesszorok SIMD (Single Instruction Multiple Data) szervezésűek, így ráadásul két *n* elemű vektor összeadásakor az összeadást utasítást csak egyszer kell felhozni, míg hagyományos skaláris feldolgozást végző processzor esetén *n*-szer.

Tömbprocesszorok

(Olvasmány: [2] 5.4. fejezet, 119-120. o.)

A vektorprocesszorokhoz hasonlóan a tömbprocesszorok is SIMD szervezésűek, de a felépítésük és a működésük teljesen más. A felépítésüket az 5.6. ábra alapján mutatjuk be.

Az utasításlehívást a tömbvezérlő egység végzi, és minden *feldolgozó és tároló egység*nek (FET) ugyanazt az utasítást küldi el, tehát a tömbprocesszor összes feldolgozó egysége ugyanazt az utasítást hajtja végre különböző adatokon. Lehetőség van feltételes utasításokra, amikben a helyileg tárolt adatok befolyásolják a feltétel kiértékelését, így lehetséges az, hogy egy utasítást némelyik FET végrehajt, másik pedig kihagy. A FET-ek a szomszédaikkal (valamilyen topológia szerint) össze vannak kötve így adatcserére képesek.

Taxonomy of Flynn

DSP (Digital Signal Processing) coprocessors improve real-time performance because they extend the instruction set to support faster, specialized instructions. These devices typically are used to extend the instruction set, and not for multiprocessing. The main processor loads certain registers with data for the coprocessor, issues an instruction starting the coprocessor, and then suspends itself until the coprocessor finishes. Usually, this involves two handshaking signals between the main processor and coprocessor.
For example, consider a typical microprocessor and associated coprocessor. Suppose there are two signals between them. When the main processor wishes to use the coprocessor, it loads global variables with the operands and sends a signal to the coprocessor. Then the main processor suspends itself. When the coprocessor finishes the operation, it places the result in another global variable, issues a signal to awaken the main processor, and suspends itself. The main processor then resumes its fetch–execute cycle.

NON-VON NEUMANN ARCHITECTURES

The limitations of the serial bus structure in most computer systems and the data intensive needs of imaging applications have led to the use of a variety of non-von Neumann-style architectures in image processing systems. A brief description of these special computing environments in the context of software engineering is valuable, because the design must be such that it exploits the advantages of the underlying hardware. In other words, while the architecture is often selected to fit the application, the application must be designed to fit the architecture.

Taxonomy of Flynn (1966).

To describe these non-von Neumann or parallel architectures, a generally accepted taxonomy is that of Flynn (1966). The classification is based on the notion of two streams of information flow to a processor: instructions and data. These two streams can be either single or multiple, giving four classes of machines:

1. Single instruction single data (SISD)
2. Single instruction multiple data (SIMD)
3. Multiple instruction single data (MISD)
4. Multiple instruction multiple data (MIMD)

Table 5.4 shows the four primary classes and some of the architectures that fit in those classes. Most of these architectures will be briefly discussed.

SINGLE INSTRUCTION SINGLE DATA

The SISD architectures encompass standard serial von Neumann architecture computers. In a sense, the SISD category is the base metric for Flynn's taxonomy.

SINGLE INSTRUCTION MULTIPLE DATA

The SIMD computers are essentially array processors. This type of parallel computer architecture has n-processors, each executing the same instruction, but on different data streams. Often each element in the *array* can only communicate with its nearest neighbour. Computer architectures that are usually classified as SIMD are the *systolic*

TABLE 5.4
Flynn's Classification Scheme for Parallel Computer Architectures

	Single Data Stream	Multiple Data Stream
Single Instruction Stream	von Neumann processors	Systolic processors
	RISC	Wave-front processors
Multiple Instruction Stream	Pipelined architectures	Data flow processors
	VLIW processors	Transputers
		Grid computers
		Multiprocessors

and wave-front array computers. In both types of processor, each processing element executes the same (and only) instruction, but on different data. Hence these architectures are SIMD. SIMD machines are widely used for such *imaging computation* as *matrix arithmetic and convolution*.

MULTIPLE INSTRUCTION SINGLE DATA

The MISD computer architecture lends itself naturally to those computations requiring an input to be subjected to several operations, each receiving the input in its original form. These applications include classification problems and *digital signal processing*. MISD architectures include *PIPELINED* and very long instruction word architectures (*VLIW*). In pipelined architectures, more than one instruction can be processed simultaneously (one for each level of pipeline). Similarly, VLIW computers tend to be implemented with microinstructions that have very long bit-lengths (and hence more capability). Thus, rather than breaking down macroinstructions into numerous microinstructions, several (nonconflicting) macroinstructions can be combined into several microinstructions.

MULTIPLE INSTRUCTION MULTIPLE DATA

MIMD computers involve large numbers of processors capable of executing more than one instruction on more than one datum at any instant. Except for networks of distributed multiprocessors working on the same problem (grid computing), these are "exotic" architectures. MIMD computers include *data flow computers, grid computers, networks of heterogeneous processors*, and *transputers*.