INFORMATIKA I.

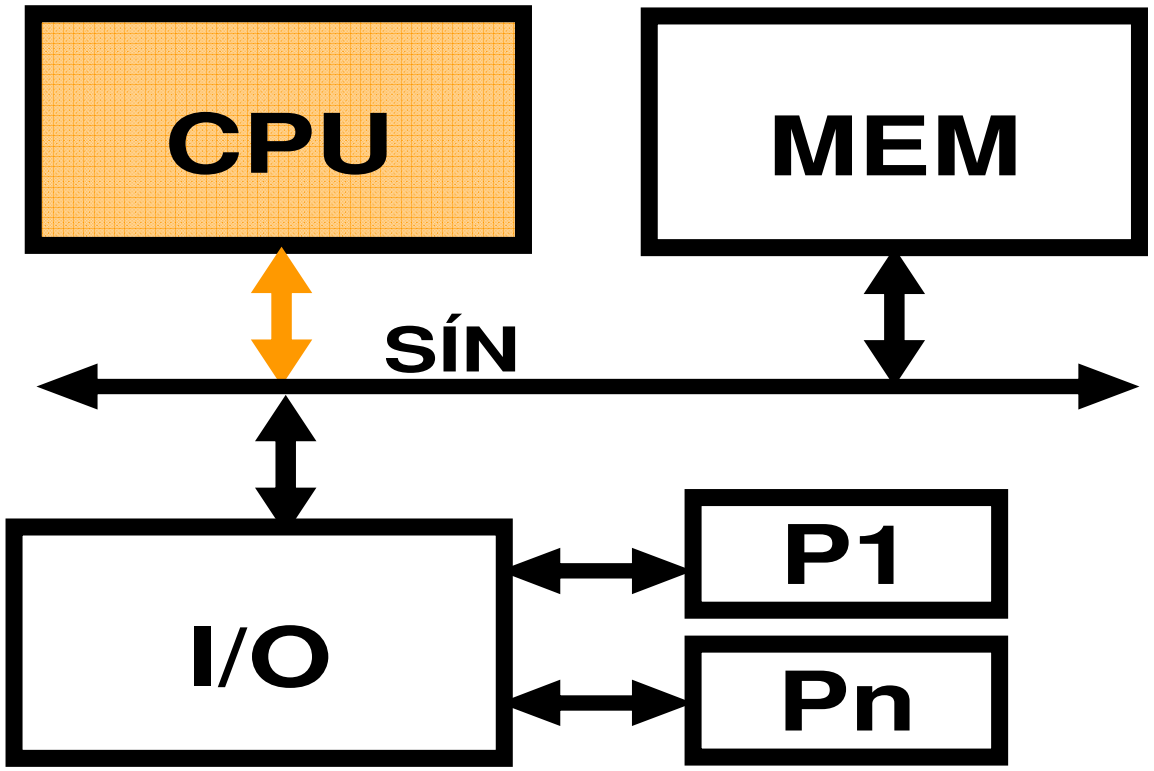
BMEVIIIAB08

Utasítás rendszer architektúra (ISA)

CPU feladata

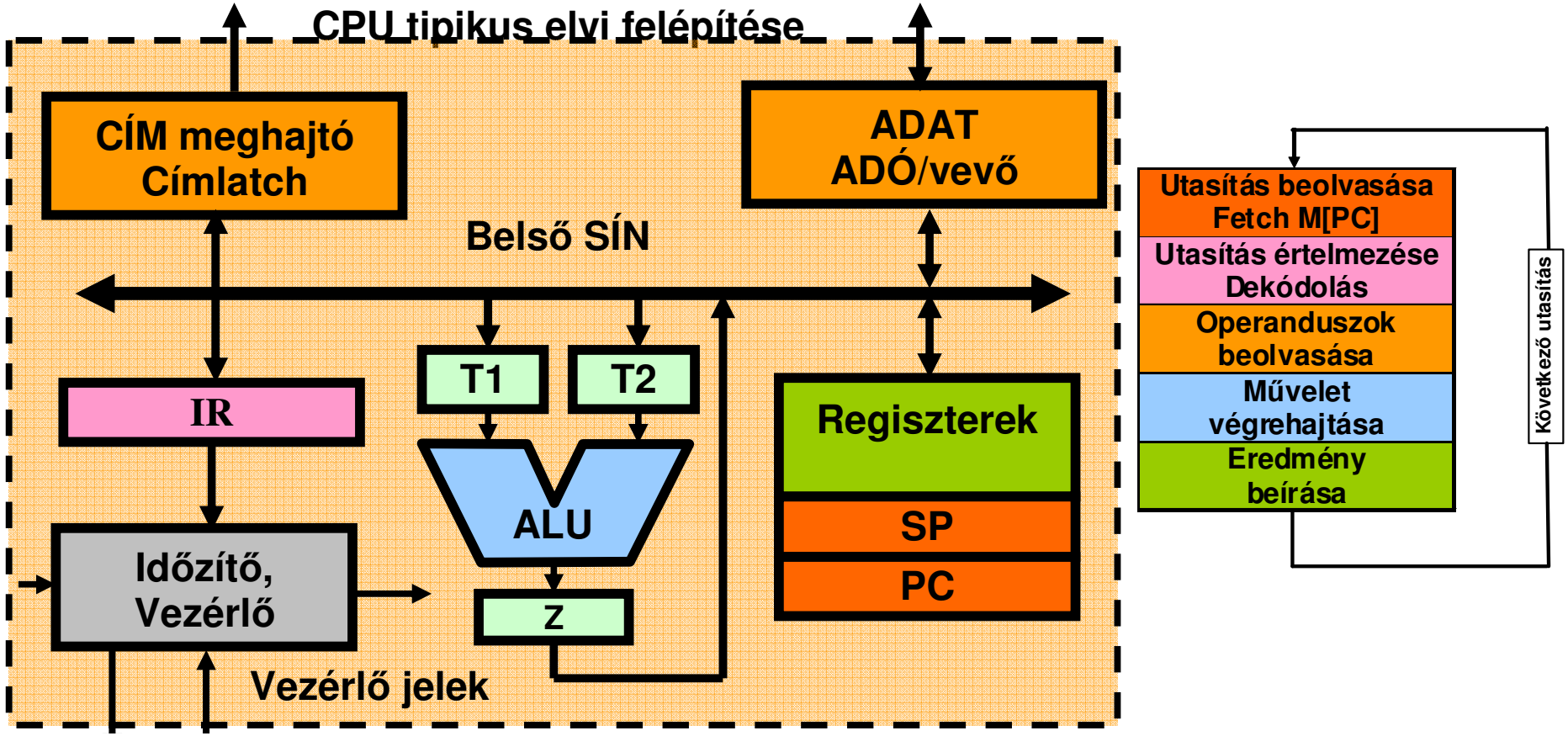
- *A tárolt program utasításainak végrehajtása*
- *A számítógép részegységei közötti adatforgalom vezérlése*

Számítógép elvi felépítése



CPU feladata

- Neumann modell szerinti szekvenciális végrehajtás
  - /elvi szintű, fizikailag nem biztos, hogy teljesül/



Utasítás rendszer architektúra (ISA)

IBM Sytem360 1964

- *Assembly programozói modell ?!* ***/ISA Instruction Set Architecture/***
- *A CPU az ISA egy megvalósításában vesz részt*
  - /A CPU az utasítás készlet megvalósítását biztosító implementáció magja/

→ *Több CPU is megvalósíthatja ugyanazt az ISA-t*

  - Pl. 360/30 8 bites belső sín, 360/60 64 bites belső sín, vagy 8086/88, x86 család

- **Utasítások**

- Művelet /***mit csinál*** /
  - *Művelet típusokat*
  - *A működés pontos leírását, használatát*
- Operandusok /***milyen adatokon*** /
  - *Adatok számát*
  - *Adat típusokat*

- Operandusok helye */**adatok helye és elérése** /*
  - *Regisztereket*
  - *Címzési módokat*
- Memóriamodell */**elhelyezés, elérhető tároló területet** /*

❖ Utasítás kialakítási szempontok

– /Több szempont hit, nézőpont (gyártó-tervező), kérdése!/

- Implementálhatóság- *egyszerűség*
  (Mindegyik implementálható)
- Ortogonalitás
- Következetesség
- Kompatibilitás
- Programozhatóság ***(Kinek a szempontjából?)***
- Magas szintű nyelvek támogatása

ISA kialakítási szempontok

Magas szintű ⟷ **Alacsony szintű**

Probléma közeli ***„emberi gondolkodás” közeli***

Utasítások

Gépi

Szemantikus rés áthidalása

Fordító-értelmező program hw-ben

Gépi utasítások szintjének növelése

- **Melyik a legjobb?**
- **Algol, Cobol, Fortran Pascal, C, C++, C#, Java, J++, stb.**

→ */JVM /*

CISC
/Complex Instruction Set Computer/
$I_N \downarrow \rightarrow f \downarrow$

Csökkentése

RISC + fordító
/Reduced Instruction Set Computer/
$I_N \uparrow \rightarrow f \uparrow$

Hatékonyság (*Kinek a szempontjából? Mérhető?)*

- *Mérő algoritmusok (Benchmark)*

$$t_{MA} = I_N(A) \cdot C_N(I) \cdot T(C)$$

$$\mathbf{P} = \frac{1}{\mathbf{t}_{M\,A}} = \frac{IPC \cdot f}{I_N(A)}$$

- I_N az utasításkészlettől függ
- *Egyszerű utasításoknál f növelhető*

❖Utasítás felépítés

Műveleti kód (*mindig van*)	Operandusok és eredmény címei	Következő utasítás címe *(elmaradhat)*

Utasításkészlet címzési típusok (*hány címes*)

- 4 címes: *két operandus -, eredmény - ,következő utasítás címe*

Műv. kód	Op1 címe	Op2 címe	eredmény címe	következő utasítás címe

  - Pl.: EDVAC *(ma csak mikroprogramozott vezérlőben)*

❑PC + vezérlés átadó utasítások bevezetése ➔3 címes

- 3 címes: *két operandus címe, eredmény címe*

Műv. kód	Op1 címe	Op2 címe	eredmény címe

  - Elsősorban RISC regiszter - referens utasításoknál
    - ❑ ***Pl.: Add R_3,R_1,R_2*** ***$R_3 \leftarrow R_1+R_2$***
      (az operandusok regiszterekben)

❑Az eredmény az egyik operandus helyére kerül ➔2 címes

- 2 címes: *két operandus címe*

Műv. kód	Op1 címe /eredmény/	Op2 címe *eredmény*

  - CISC-nél elterjedt *(IBM360, PDP 11, VAX)*
    - ***AddB X, Y ;*** ***M(Y) ← M(X)+M(Y) byte***
    - ***AddW X, Y ;*** ***M(Y) ← M(X)+M(Y) word***
    - ***AddDW X, Y ;*** ***M(Y) ← M(X)+M(Y) double word***
    - ***Add*** R_1, R_2 ***;*** $R_2 \leftarrow R_1+R_2$

- Egy operandus kijelölt helyre pl.: AC ➔ 1 címes

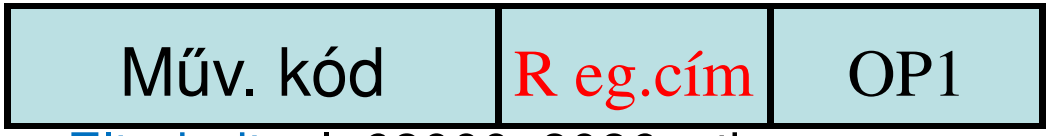
- 1 címes: egy operandus címe

Műv. kód	Op1 címe

  - ***Add X*** ***Acc ← Acc+M(X)***
  - Korai és a kis teljesítményű rendszereknél *(IBM 650, 8085)*

❑Az egyik operandus egy regiszterben ➔ 1.5 címes

- 1.5 címes: *egy operandus címe, Regiszter címe* ***(kevesebb bit!)***



  – Elterjedt, pl.:68000, 8086, stb.

❑ ***Add D_1, elsoszam***

❑Minek cím? ➔0 címes

- 0 címes: *operandus fix helyen van /verem teteje/*

  – B5000(1960) ALGOL nyelvre,
    (a részben a CPU-ban felépített stack miatt volt gyorsabb)
  – Sun picoJava I-II (1998) JVM –re

- Utasítások száma

$$Z = X + Y$$

Utasítások száma különböző címszámú utasításokkal				
3 címes RISC Reg- Reg	*2 címes CISC Mem- Mem*	*1.5 címes CISC Reg-Mem*	*1 címes ACC-Mem*	*0 címes Stack*
Load R_1, X *Load R_2,Y* *AddR_3,R_1,R_2* *Store Z,R_3*	*Mov Z, X* *Add Z, Y*	*Load R_1, X* *Add R_1, Y* *Store Z,R_1*	*Load X* *Add Y* *Store Z*	*Push X* *Push Y* *Add* *Pop Z*
Megjegyzés: utasítások száma nem egyenlő a memóriához fordulások számával!?				

Utasítások művelet szerinti csoportosítása

- Adatok kezelése
  - Adattípusok
    - Bit, byte,…, BCD, integer, FP, karakter, - láncok, stb.
  - Adatmozgatás
    - Move, exchange, load/store, pop/push, I/O*(akár tömböt is)*
  - Műveletek adatokon
    - Aritmetikai (ADD, SUB, MUL, DIV, INC, DEC, COMP)
    - Logikai (AND, OR, NOR, ...)
    - Logikai/aritmetikai biteltolás
      - Léptetés, többszörös léptetés, forgatás
    - Karakterlánc,- bitcsoportok feldolgozása
    - Adatsorozat feldolgozása

      –MMX (Multi Media eXtension), SSE2 (Streaming SIMD Extension)
      pl.: 128 bit 4x32 bites adaton SIMD)

- Különböző adatformátum konverziók
  - Egész *(pl.:16, 32, 64, 128 bit)*
  - Lebegőpontos *(pl. 80 bit, 16, 32, 64,128 bit)*
  - BCD, csomagolt BCD, karakter, bit-, karakterlánc, stb.

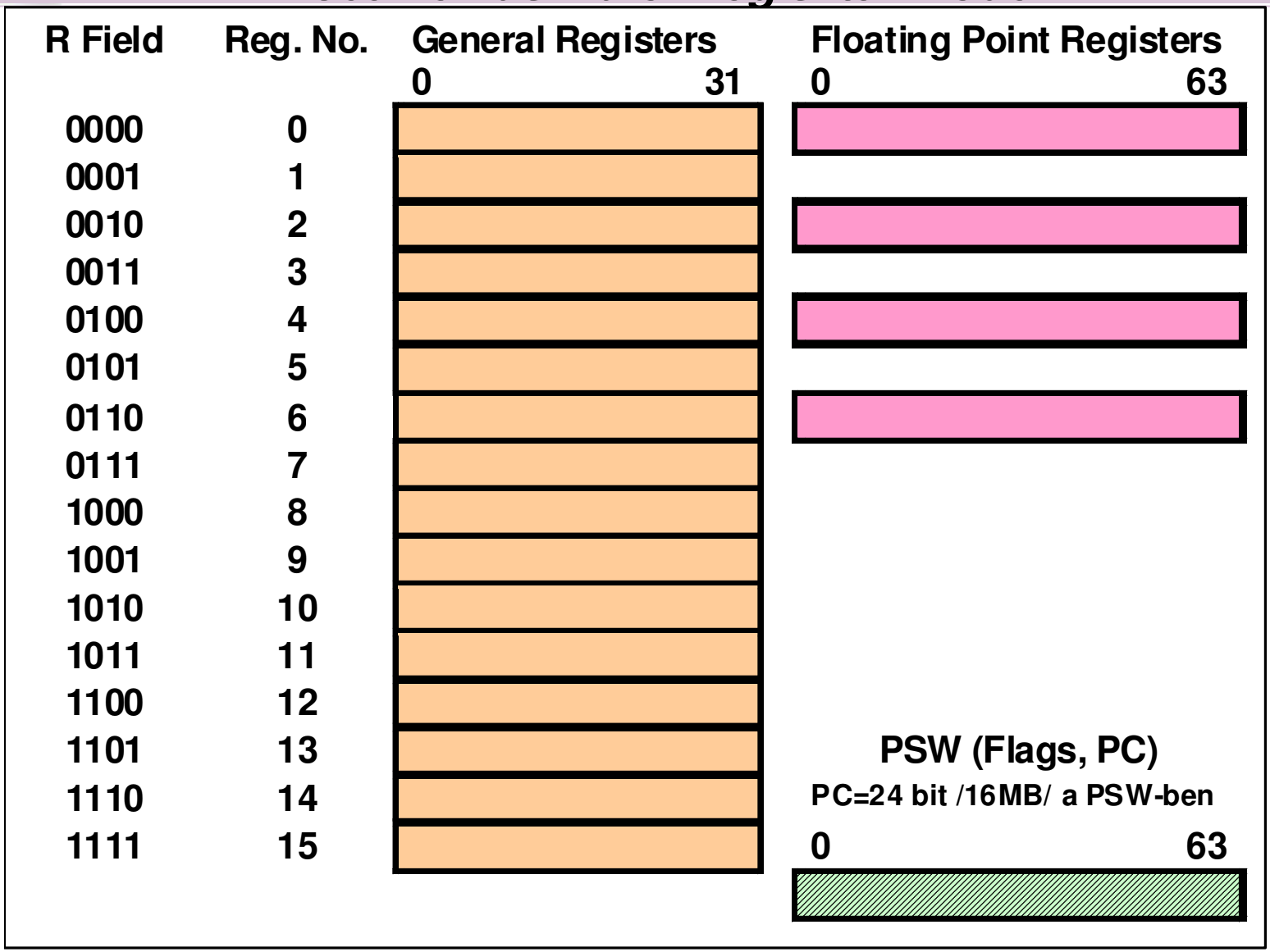
- Vezérlés, vezérlés átadás
  - Feltételes/feltétel nélküli
    - Ugrás
    - Szubrutinhívás
    - Iteráció *(hurok)*
    - Egyéb
      - Halt, Wait, Lock, stb.
      - Rendszervezérlő (*üzemmód, - címzési struktúra, - védelem állító*)

- Megszakítás tiltás/engedélyezés
- Eltérülés/csapda ***/eltér a megszakítás kezeléstő/***

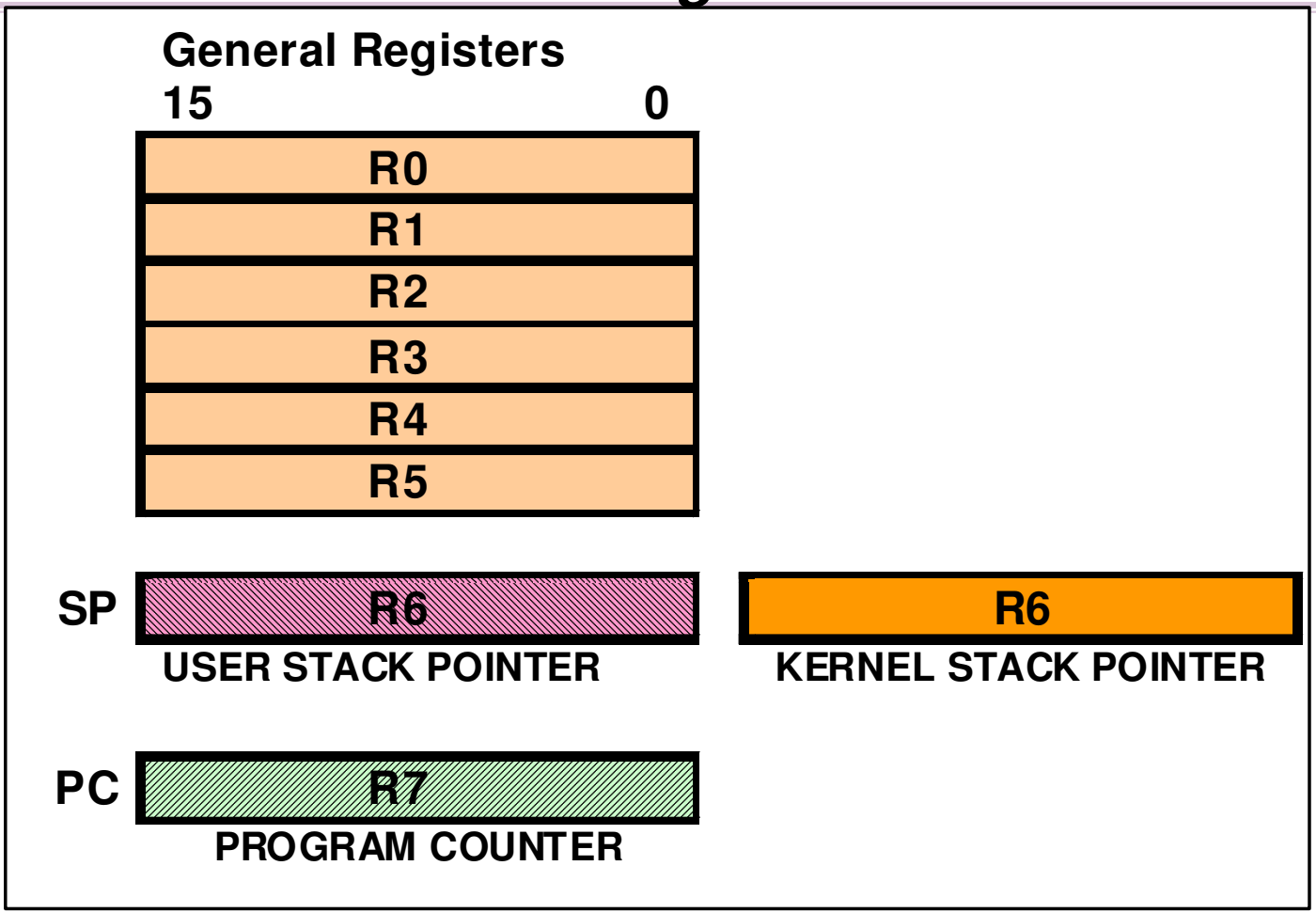
- Operandusok helye
  - Memória
  - Regiszterek
    - Regiszterek száma (8, 16, 32, 128, …) ***/Leggyorsabb elérés/***
    - Regiszter használat filozófiája
    - Általános vagy meghatározott (elrendelt) célú használat (Motorola←→Intel)
      - Felhasználás szerint
    - Cím általános, vagy cél (PC, SP)
    - Adat különböző formátum „szélesség?”
    - Speciális PSW, rendszervezérlő regiszterek, MMU, Cache, stb.
      (felhasználói programozási modellben nem hozzáférhetők)

Felhasználói Regiszterkép */ESETANULMÁNYOK/*

IBM 360 Felhasználói Regiszter modell



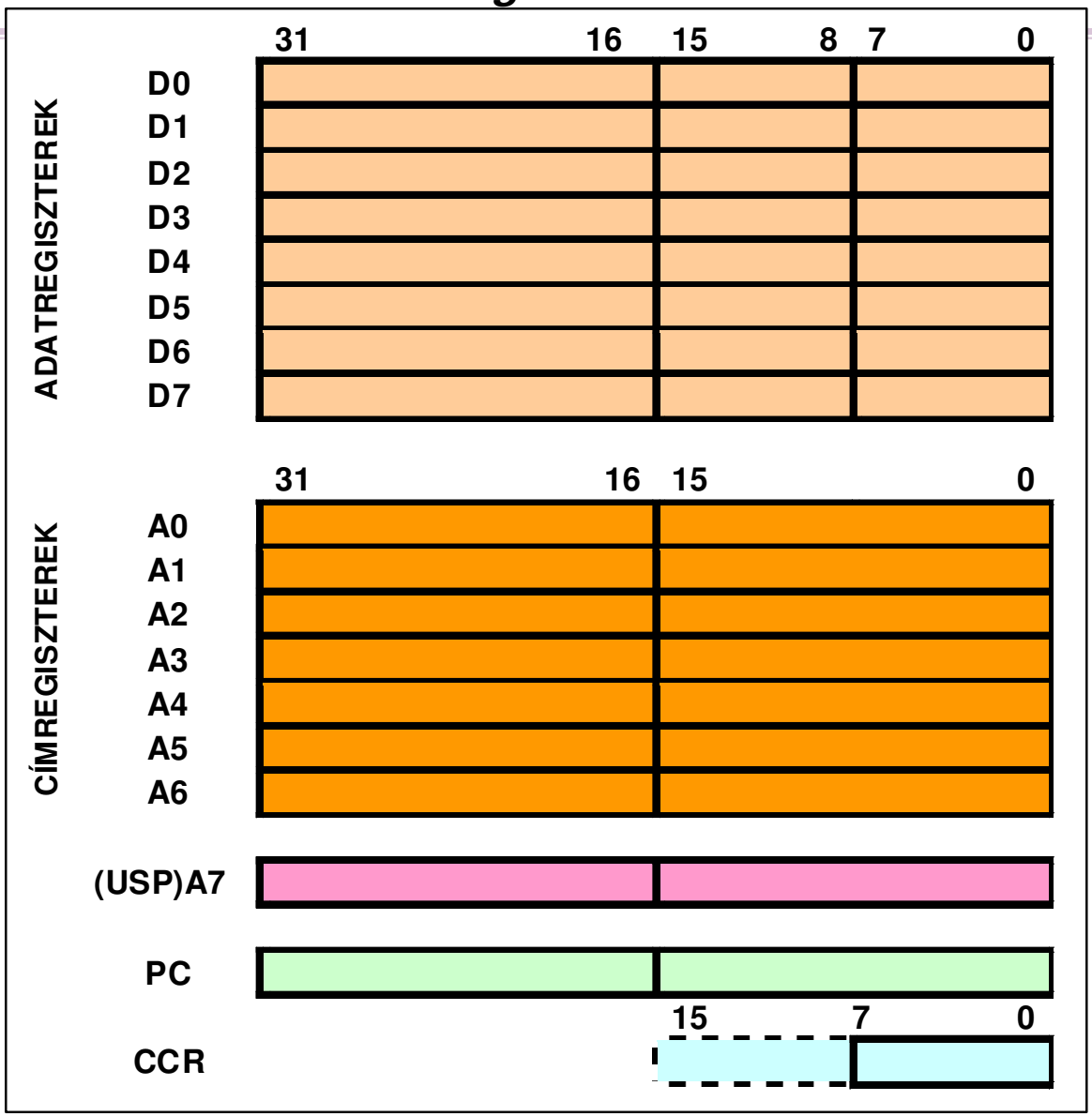
PDP 11 Felhasználói Regiszter modell



- Opcionálisan FLPA egység, FLPA utasítások, FLPA regiszterekre stack hivatkozás

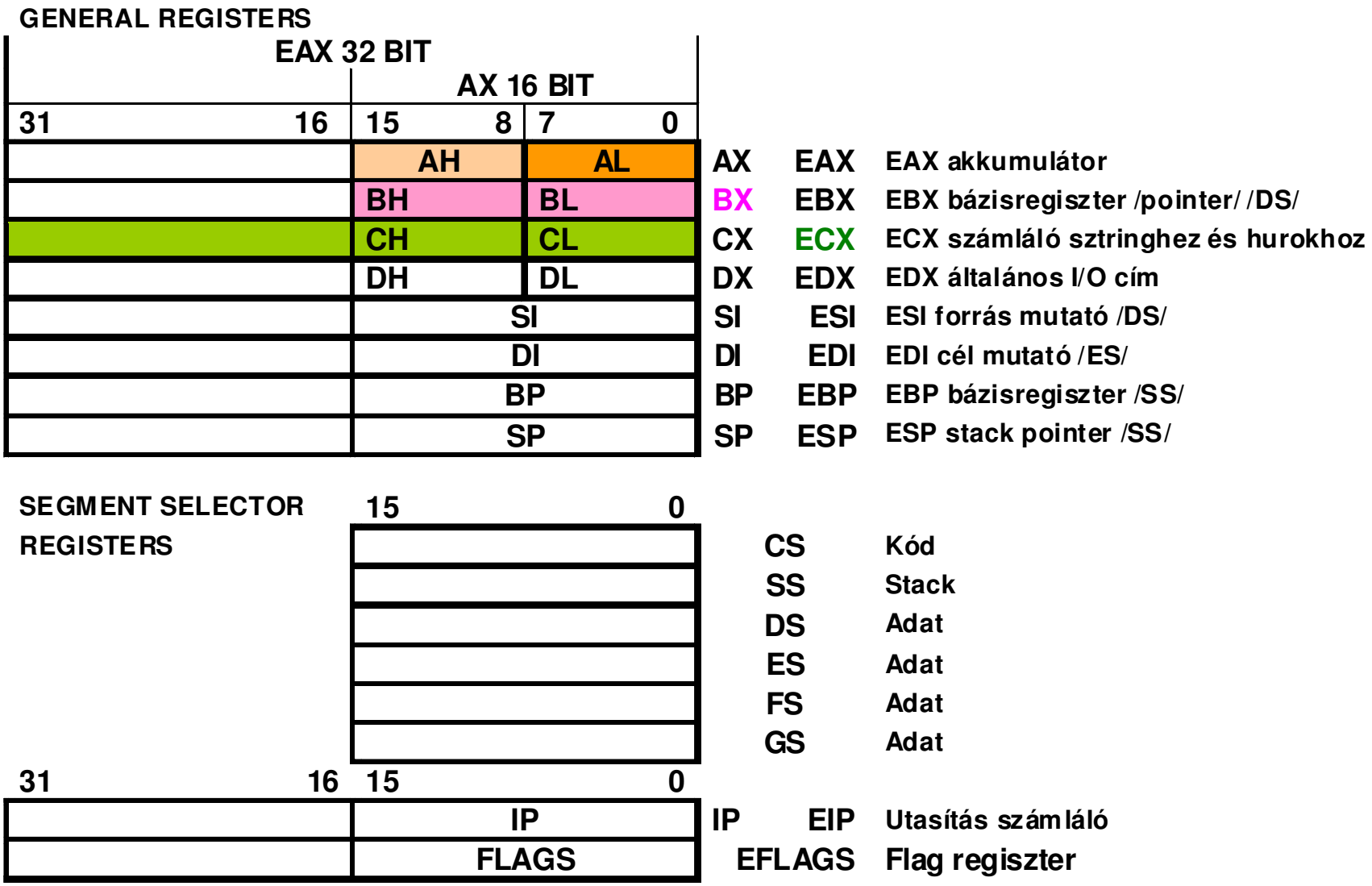
Felhasználói Regiszterkép /ESETANULMÁNYOK/

M68030 felhasználói Regiszter modell

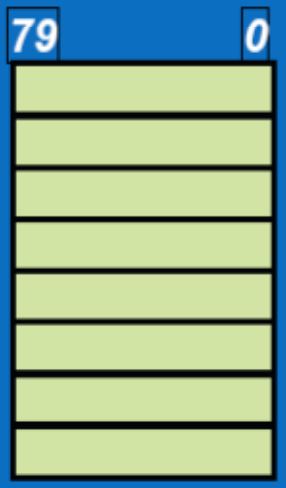


Felhasználói Regiszterkép /ESETANULMÁNYOK/

80386 Felhasználói Regiszter modell /ISA IA-32/



Registers : Extensions and Additions



X87/ MMX

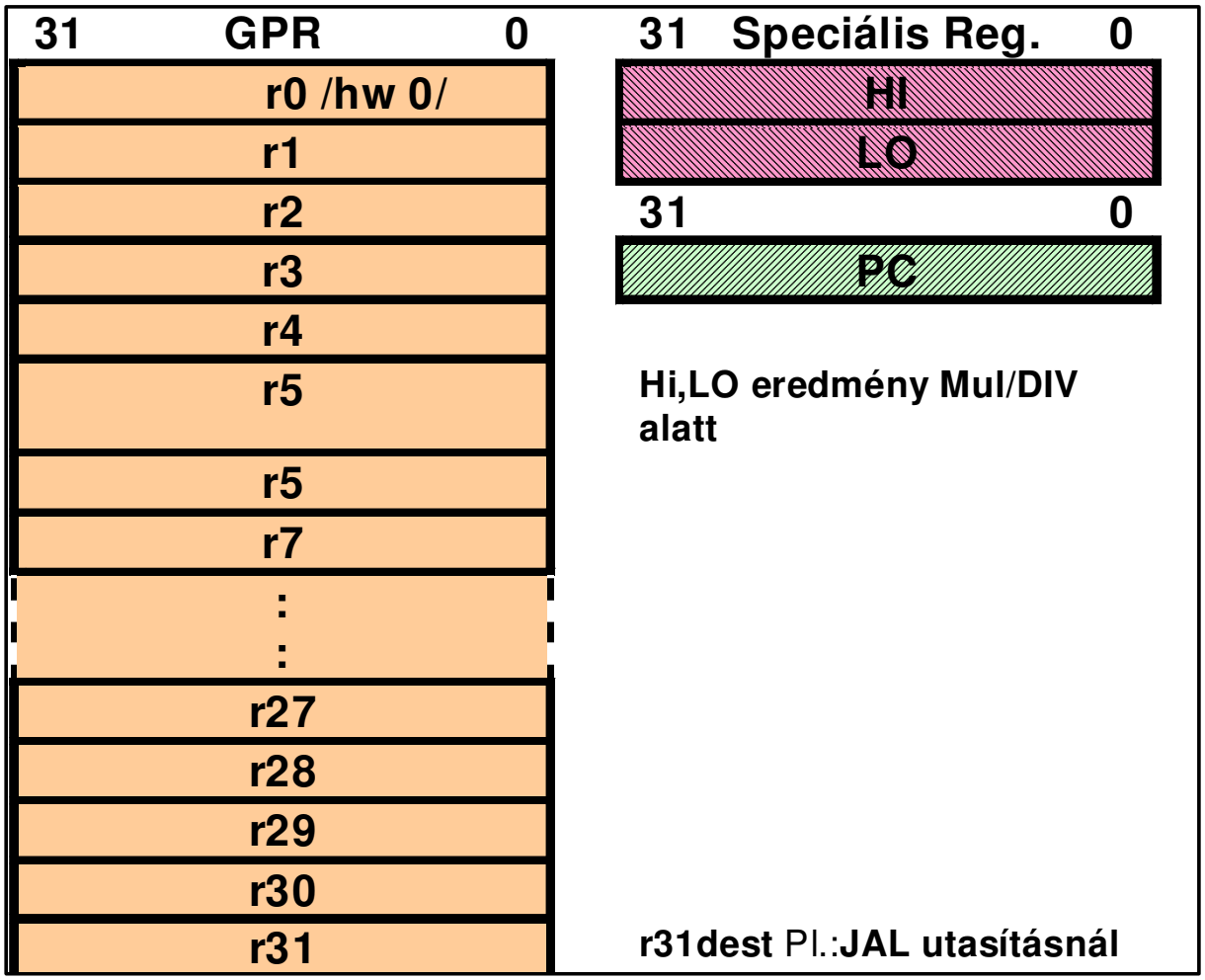
63 32	31 0
RAX	EAX
RBX	EBX
RCX	ECX
RDX	EDX
RBP	EBP
RSI	ESI
RDI	EDI
RSP	ESP
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	

RIP	EIP

127 64	63 0
XMM0	
XMM1	
XMM2	
XMM3	
XMM4	
XMM5	
XMM6	
XMM7	
XMM8	
XMM9	
XMM10	
XMM11	
XMM12	
XMM13	
XMM14	
XMM15	

Software-Visible Register or Data Structure	Legacy and Compatibility Modes			64-Bit Mode		
	Name	Number	Size (bits)	Name	Number	Size (bits)
General-Purpose Registers	EAX, EBX, ECX, EDX, EBP, ESI, EDI, ESP	8	32	RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8-15	16	64
Floating-Point Registers[1]	ST0-7	8	80	ST0-7	8	80
Multimedia Extension Registers[1]	MM0-7	8	64	MM0-7	8	64
Streaming SIMD Extension Registers	XMM0-7	8	128	XMM0-15	16	128
Instruction Pointer	EIP	1	32	RIP	1	64
Flags	EFLAGS	1	32	EFLAGS	1	32
Stack Width	—		16 or 32	—		64

MIPS-32 Felhasználói Regiszter modell /1981 John L. Hennessy, Stanford University/



(MIPS:Microprocessor without Interlocked Pipeline Stages RISC model)

Felhasználói Regiszterkép /ESETANULMÁNYOK/

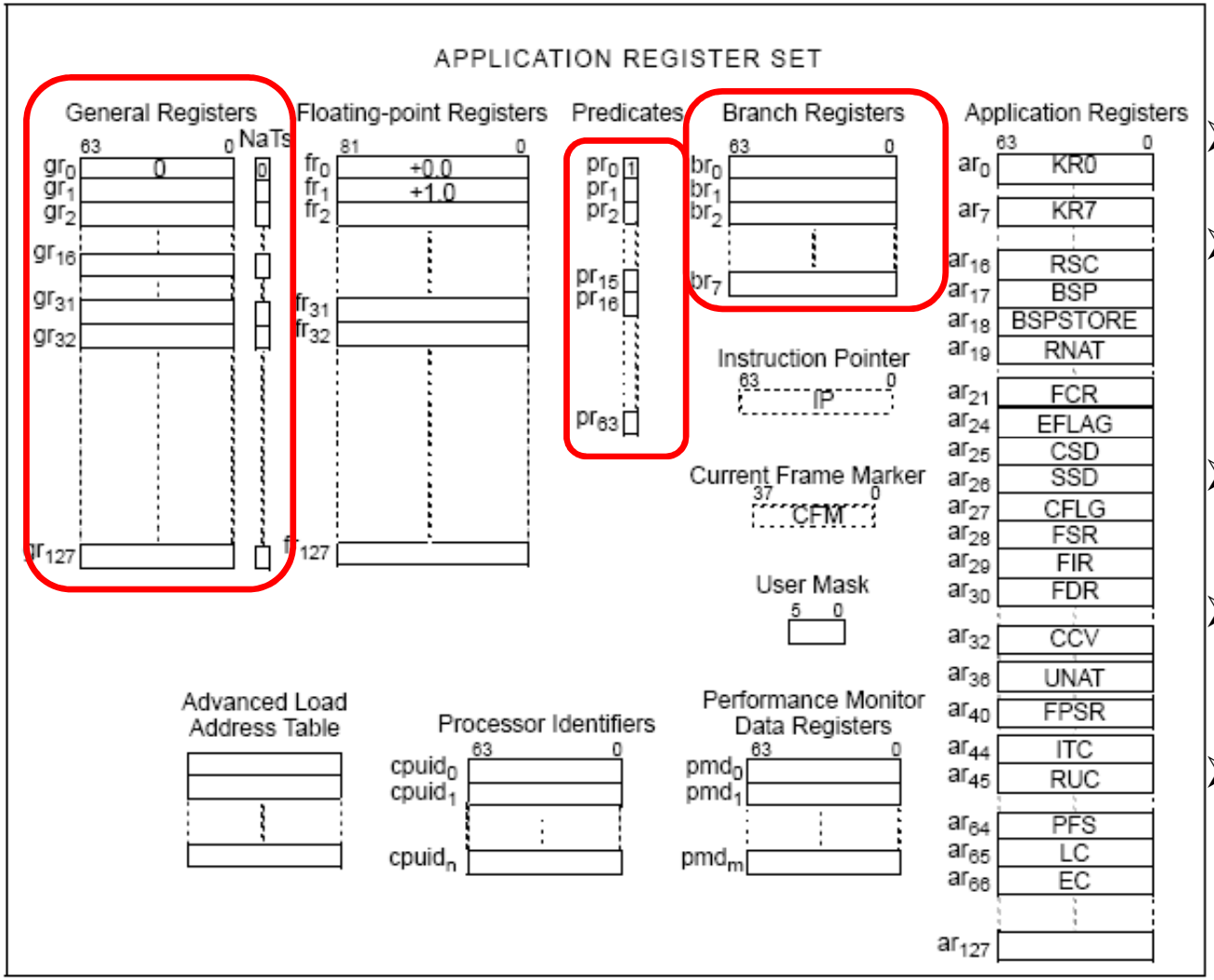
MIPS regiszter kiosztás

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

- ***Sok regiszter,* korlátozások**
- **Az *egyszerűbb* könnyebben használható**
- ***RISC II 138 regiszter*** */ablaktechnika lsd. később/*
  /RISC-I David A. Patterson Berkely/

Felhasználói Regiszterkép /ESETANULMÁNYOK/

Intel ITANIUM Felhasználói Regiszter modell



- ***GR0-GR31***
  Statikus regiszterek
  „mindig láthatók”
- ***GR8- GR31***
  IA-32 üzemmódhoz
- ***GR32-GR127***
  általános regiszter
  /regiszter stack-programozható méretű rész--ablak technika/
- ***NaTs*** */Not a thing/*
  Spekulatív sorrendű utasítás végrehajtáshoz
- ***Predikátum bitek a feltételes utasítás végrehajtáshoz***
- ***Indirekt ugrást segítő regiszterek***

Memória - elérési konvenciók

Little endian *(PDP, VAX, Intel)* ***kisebb címen kisebb helyérték***

Regiszter

31 3.byte 24	23 2.byte 16	15 1.byte 8	7 0.byte 0	REG.
00000000	00000000	00000111	11100000	2016

Memória

11	10	01	00	MEM.
3.byte	2.byte	1.byte	0.byte	
00000000	00000000	00000111	11100000	2016
31 Dupla Szó 0				32
15 szó		15 szó 0		28
7 byte 0	7 byte 0	7 byte 0	7 byte 0	24
		2/4 dsz		20
2/4 dsz			1/2sz	16
1/2sz	sz			12
	3/4 dsz			8
1/4 dsz			1/4dsz	4
3/4 dsz				0

Nem illesztett elhelyezések

Illesztett elérés → szó :kettővel maradék nélkül osztható /páros/ címen

→ dupla szó: néggyel maradék nélkül osztható címen

Big endian *(IBM 360, M680xx)* ***kisebb címen nagyobb helyérték***

Regiszter

REG.	31 3.byte 24	23 2.byte 16	15 1.byte 8	7 0.byte 0
	00000000	00000000	00000111	11100000

Memória

MEM.	00	01	10	11
	0.byte	1.byte	2.byte	3.byte
2016	00000000	00000000	00000111	11100000
32	31 Dupla szó 0			
28	15 Szó 0		15 Szó 0	
24	7 byte 0	7 byte 0	7 byte 0	7 byte 0
20	2/4 dsz			
16	1/2 sz		2/4 dsz	
12		sz		1/2sz
8	3/4 sz			
4	1/4 dsz			1/4 dsz
0		3/4 dsz		

Nem illesztett elhelyezések

- ***A nem illesztett egységeket rendszerenként eltérően kezelik***

  Intel megengedi a kompatibilitás miatt, de fizikailag processzortól függően kezeli, 8088/8086/80286 fizikailag bontja !! ***bonyolult és lassú!!***

  A Pentium /64bit/, ha kell többet olvas és belül összeállítja ***!!lassú!!***

  IBM System360, MIPS, Motorola nem engedi meg
  Az Itanium az adatot mindkét módszerrel elérheti

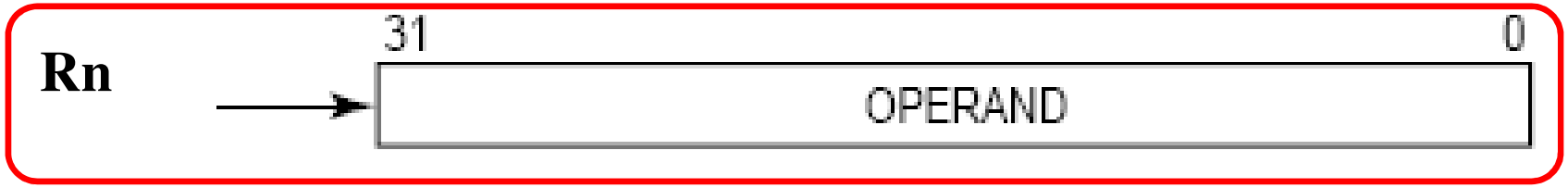
Címzési módok

- Egykomponensű cím
  (*az effektív cím /EA/ egy összetevőből áll*)
  - Közvetlen */direkt/ cím*
    - ❑ **Regiszterkijelölés** ***ADD D2,D1***

EA=Rn

Assembler

syntax: Rn



An *Címregiszter*
Dn *adatregiszter*
Rn *bármelyik /Rn helyett néha Xn Assembler szintaxis? /*
X *memória cím*

Címzési módok/ Egykomponensű cím

- Közvetlen adat /immediate/ ***/PC címzi az operandust /***

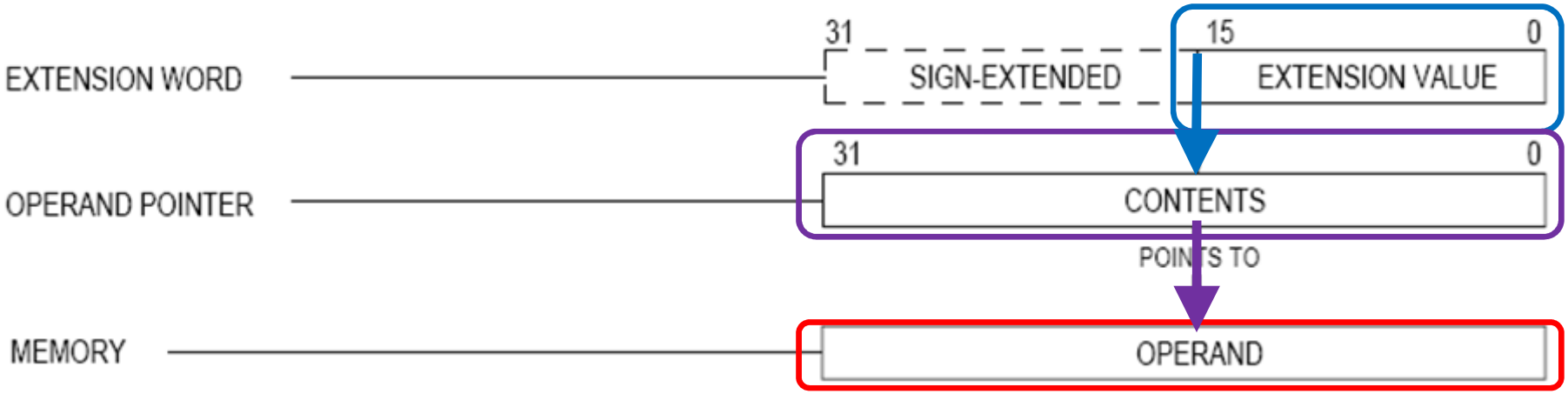
  MOVEA.W #7,A2 ***(A2=7 (16/32, itt 16 bit➔32) Motorola szintaktika)***

- Közvetlen memóriacím /*abszolút adatcím*/
  /cím az utasítás után/

LEA (Kezd).W,A1 *(az utasítás kiterjesztésben adott "Kezd" címen van az operandus (32bit))* **egyszerű**

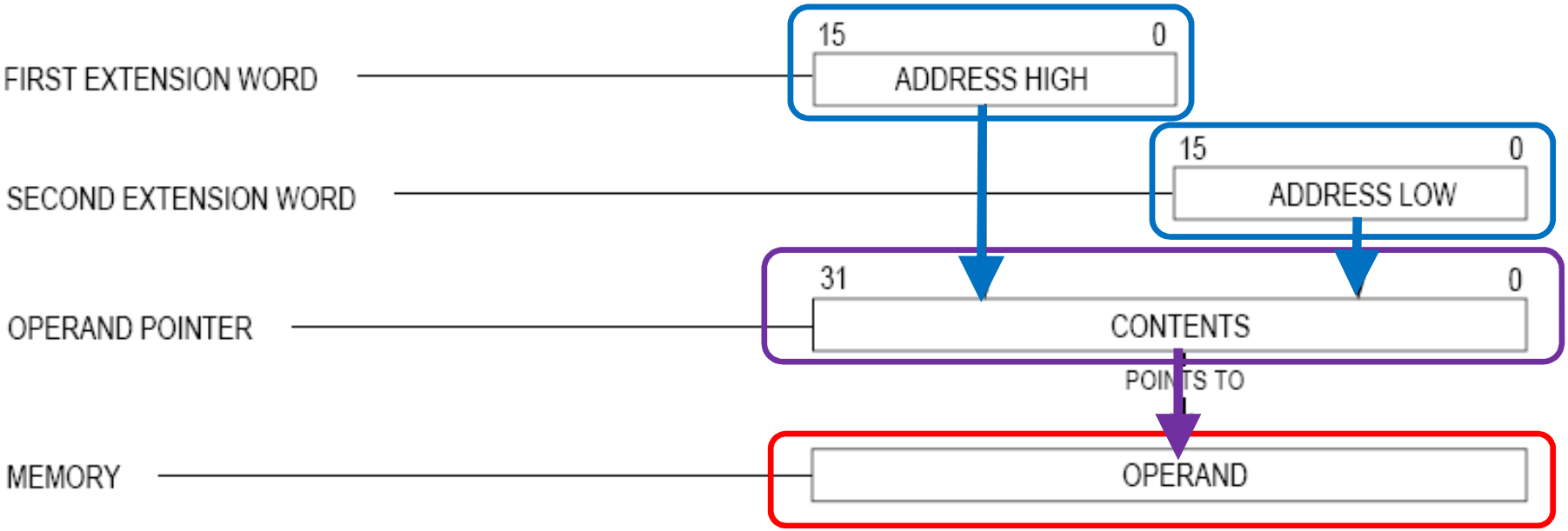
LEA (Kezd).W,A1 → *(rövid cím)*

GENERATION:	EA GIVEN
ASSEMBLER SYNTAX:	(xxx).W
EA MODE FIELD:	111
EA REGISTER FIELD:	000
NUMBER OF EXTENSION WORDS:	1



MOVE.W D0,(XX).L ☞ *(hosszú cím)*

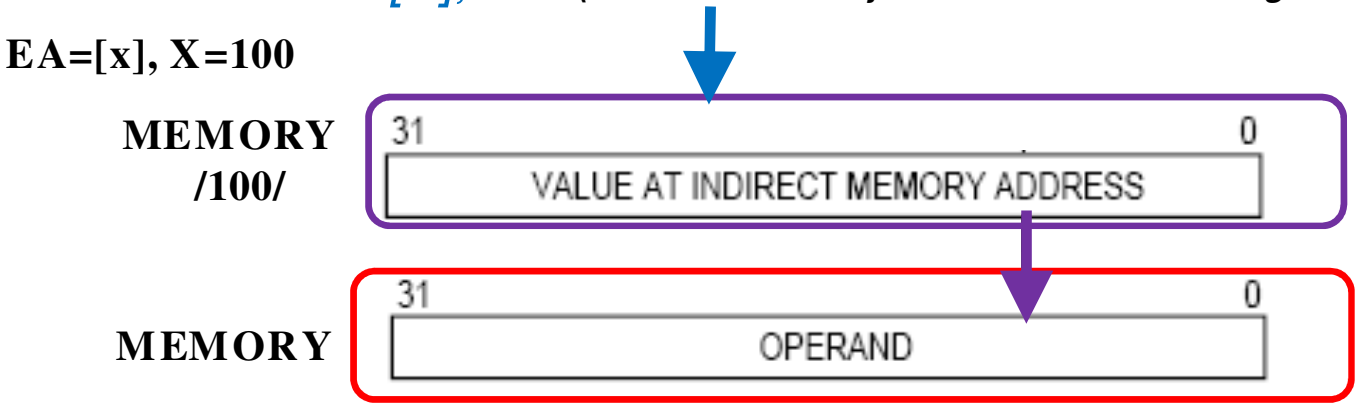
GENERATION:	EA GIVEN
ASSEMBLER SYNTAX:	(xxx).L
EA MODE FIELD:	111
EA REGISTER FIELD:	001
NUMBER OF EXTENSION WORDS:	2



• Indirekt /*közvetett*/ cím

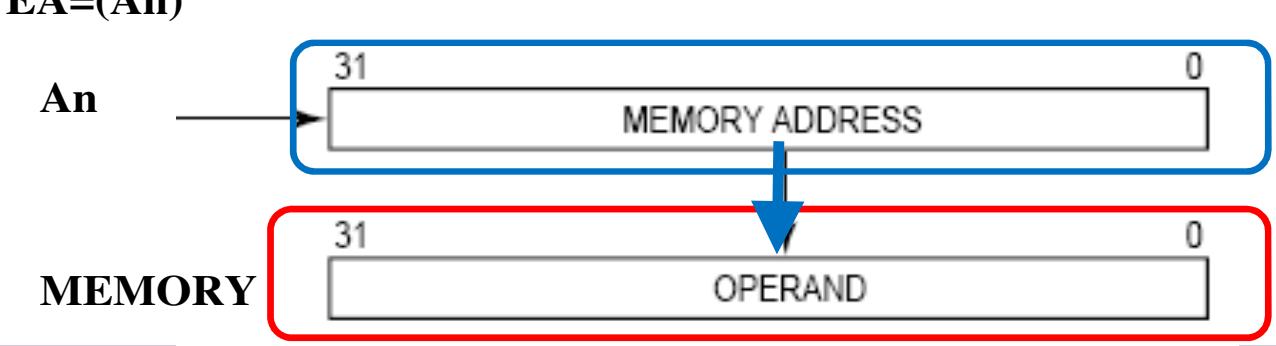
❑ Memória indirekt cím: *utasítás címrésze, a címet tartalmazó memóriarekeszre utal*

MOVEA.L [X],A3 ***(az ábra nem mutatja a 100 című memória megcímzését)***



❑ Regiszter indirekt címzés

MOVE.L #6 (A5) ***MOVE.W (A1),D0***



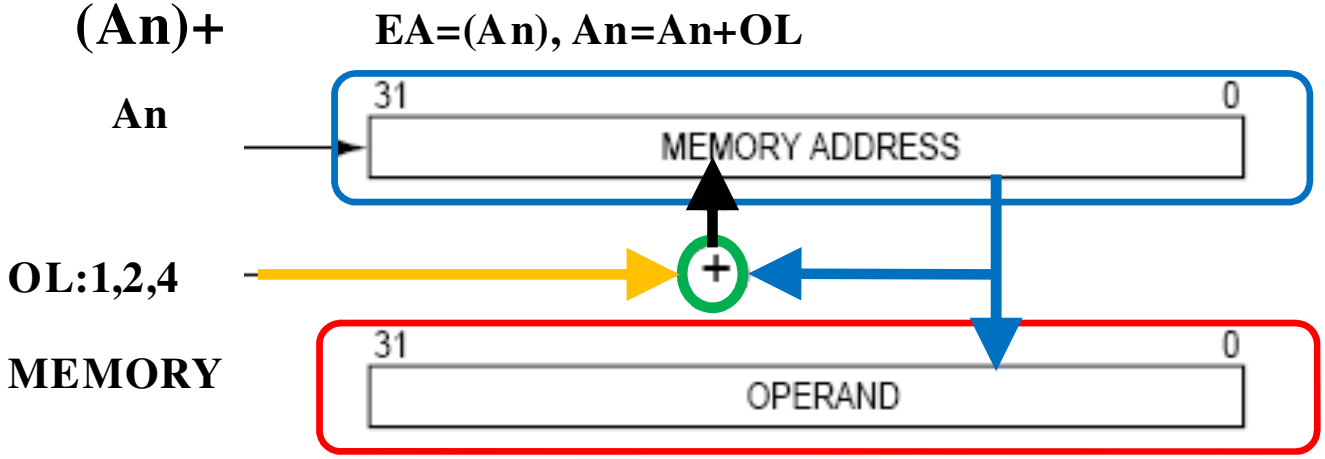
- Automatikus címmódosítás */regiszter indirekt/*

  {pre | post} auto { dekremens | inkremens }

  / - (An), (An) +/

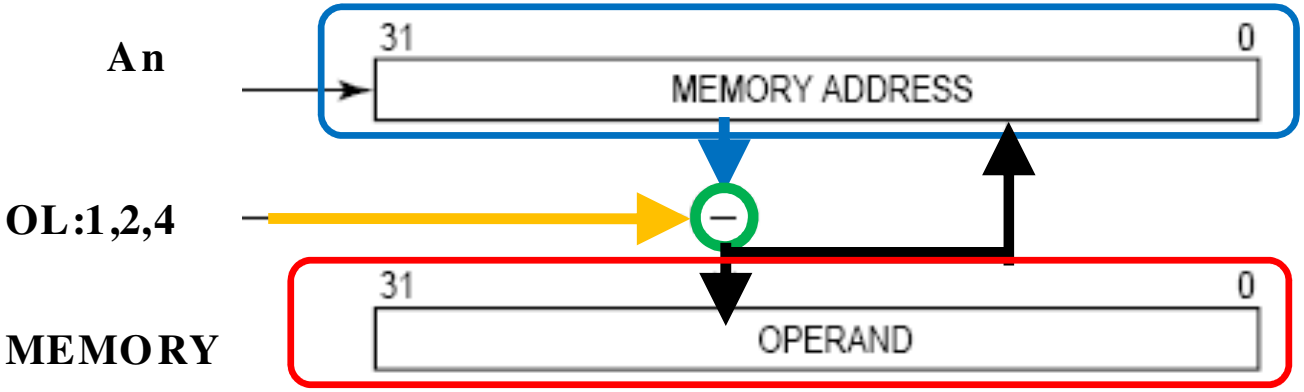
ADD.L (A3)+,D1 MOVE.W D1,-(A4) MOVE.B (A4)+,D1

- *POP, PUSH kiváltható, illetve programozottan több verem is kialakítható*



-(An)

An=An - OL, EA=(An),



- Magába foglalt */implicit/* címzés

  /az utasítás nem tartalmaz címrészt a cím a művelthez rendelten HW-ben képződik/

  pl.: veremműveleteknél

  PEA (link); SP=SP-4, (SP)=(LINK) /EA/

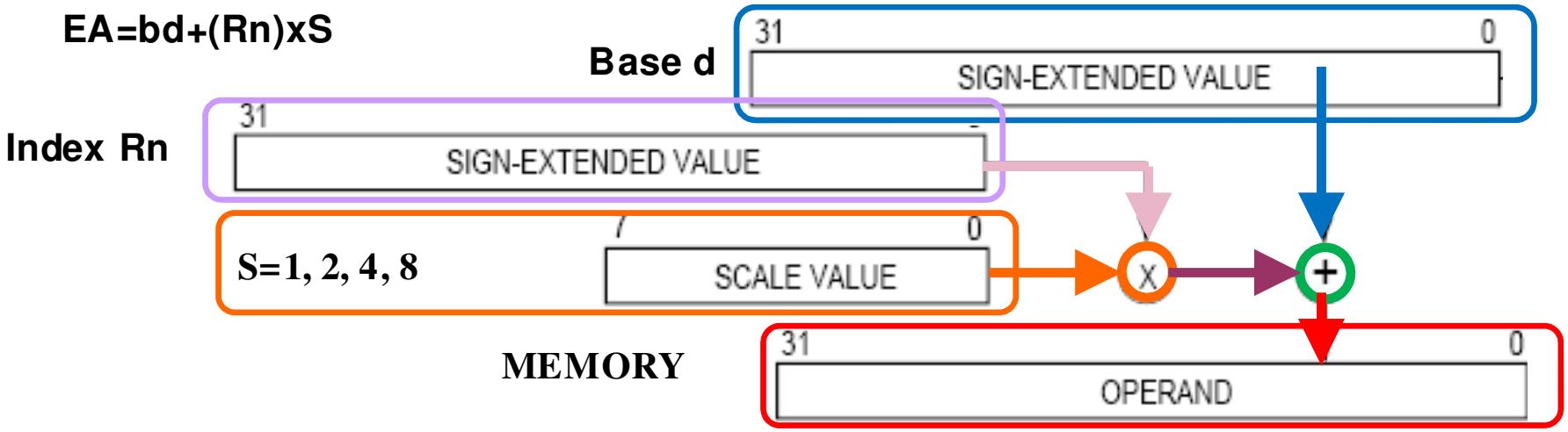
- Kétkomponensű cím */az effektív cím (EA) két összetevőből áll/*
  - Indexelt címzés */Az utasítás címrésze* ***bázis címet*** */Bd/*
    és egy regiszter kijelölést tartalmaz

EA a Bd és egy indexregiszter tartalmából képzett értéknek az összege

EA = báziscím + ((indexregiszter) * méret)

a méret (S) lehet 1,2,4,8 ➔ tömbkezeléshez előnyös

*MOVE.L (tomb,A1.L*4),D0* ***EA = tomb+((A1)x4)***



Ci=Ai+Bi kiszámítása egykomponensű és indexregiszteres címzéssel

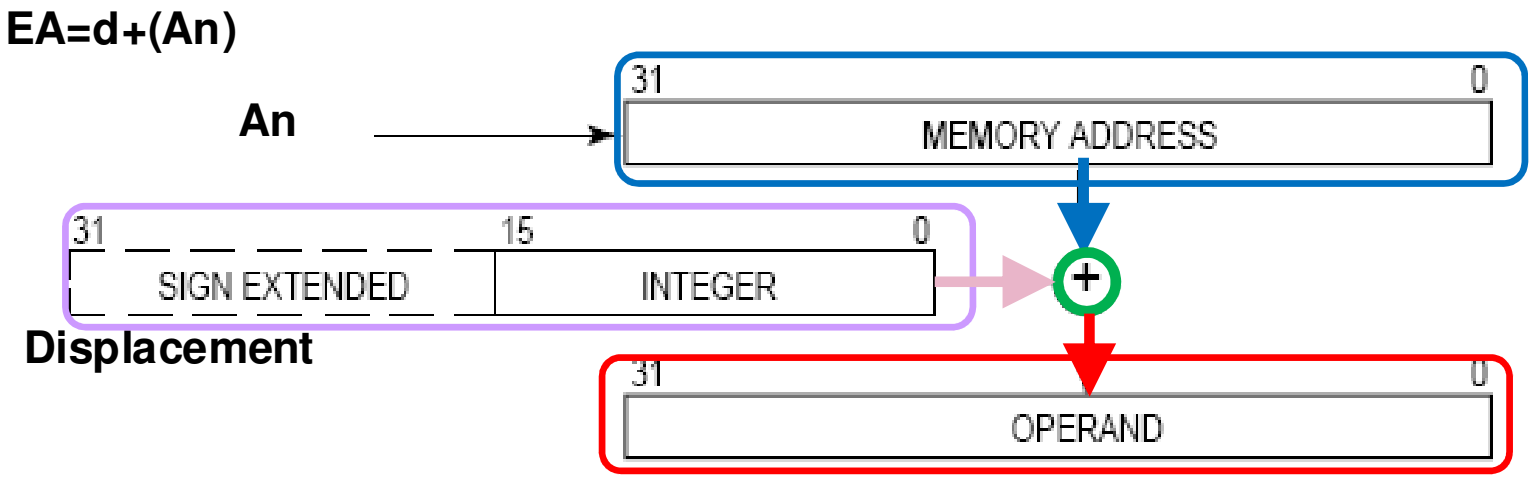
- Bázis relatív címzés

  Az utasítás címrésze ***eltolást*** ***(offset/displacement)*** *tartalmaz*

EA az eltolásnak és egy bázisregiszter tartalmának az összege

EA = (bázisregiszter) + displacement

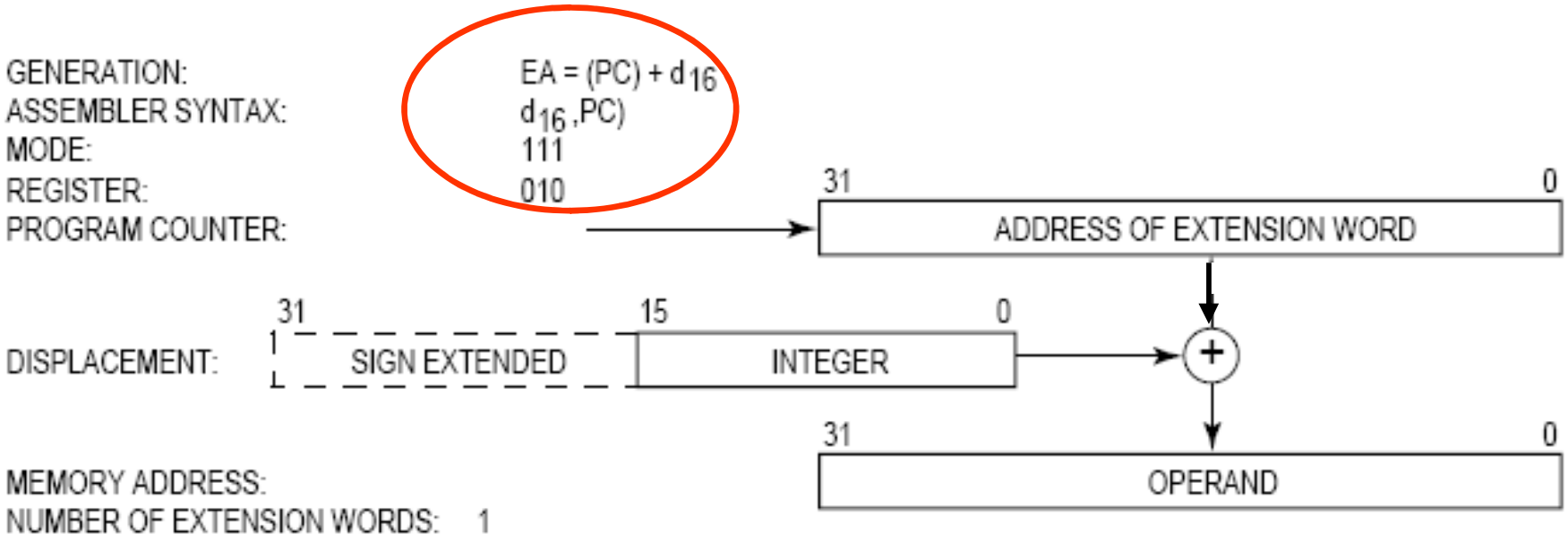
➔ relokációnál előnyös



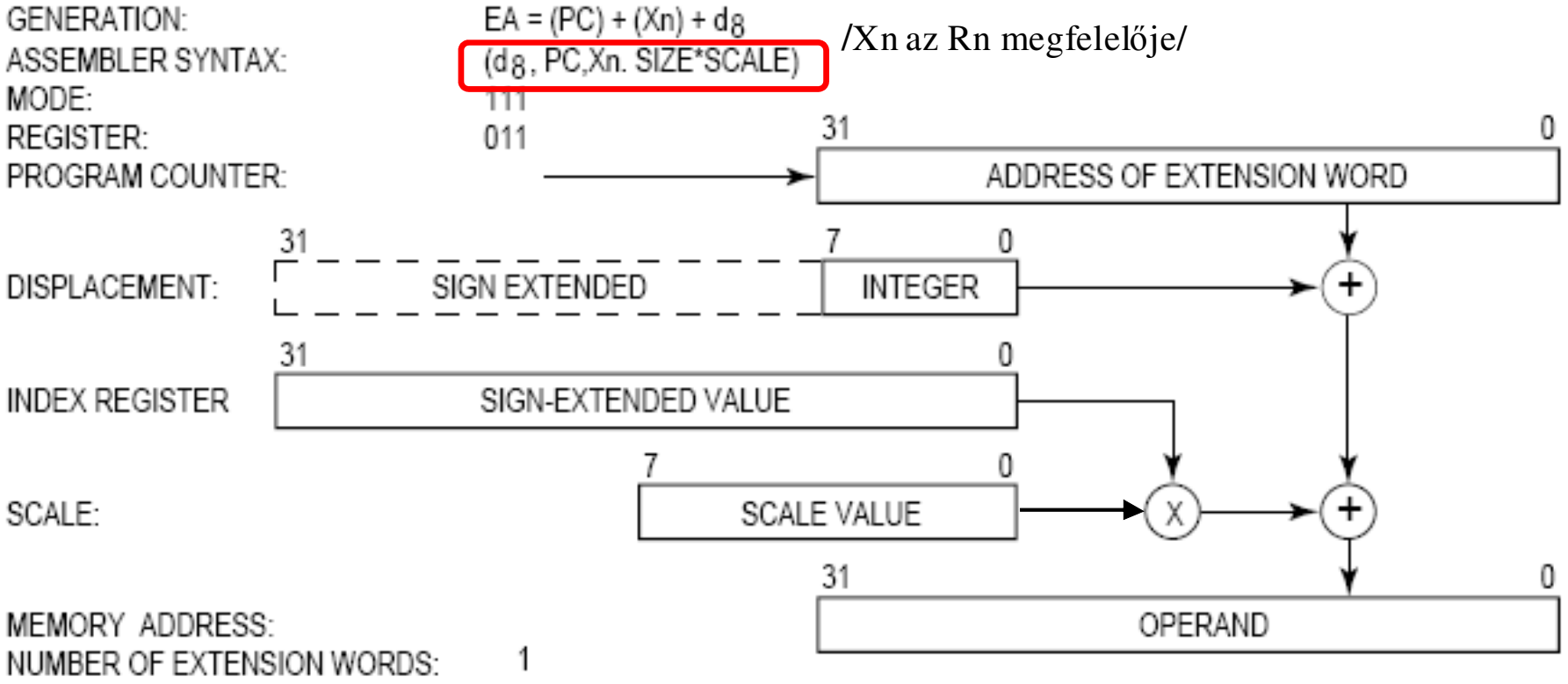
- Programszámláló - relatív címzés

  EA az utasításban szereplő címrész /eltolás (displacement)/ és a PC tartalmának előjel-helyes összegeként áll elő

  BRA LOOP /Ea=(PC)+#disp._LOOP/



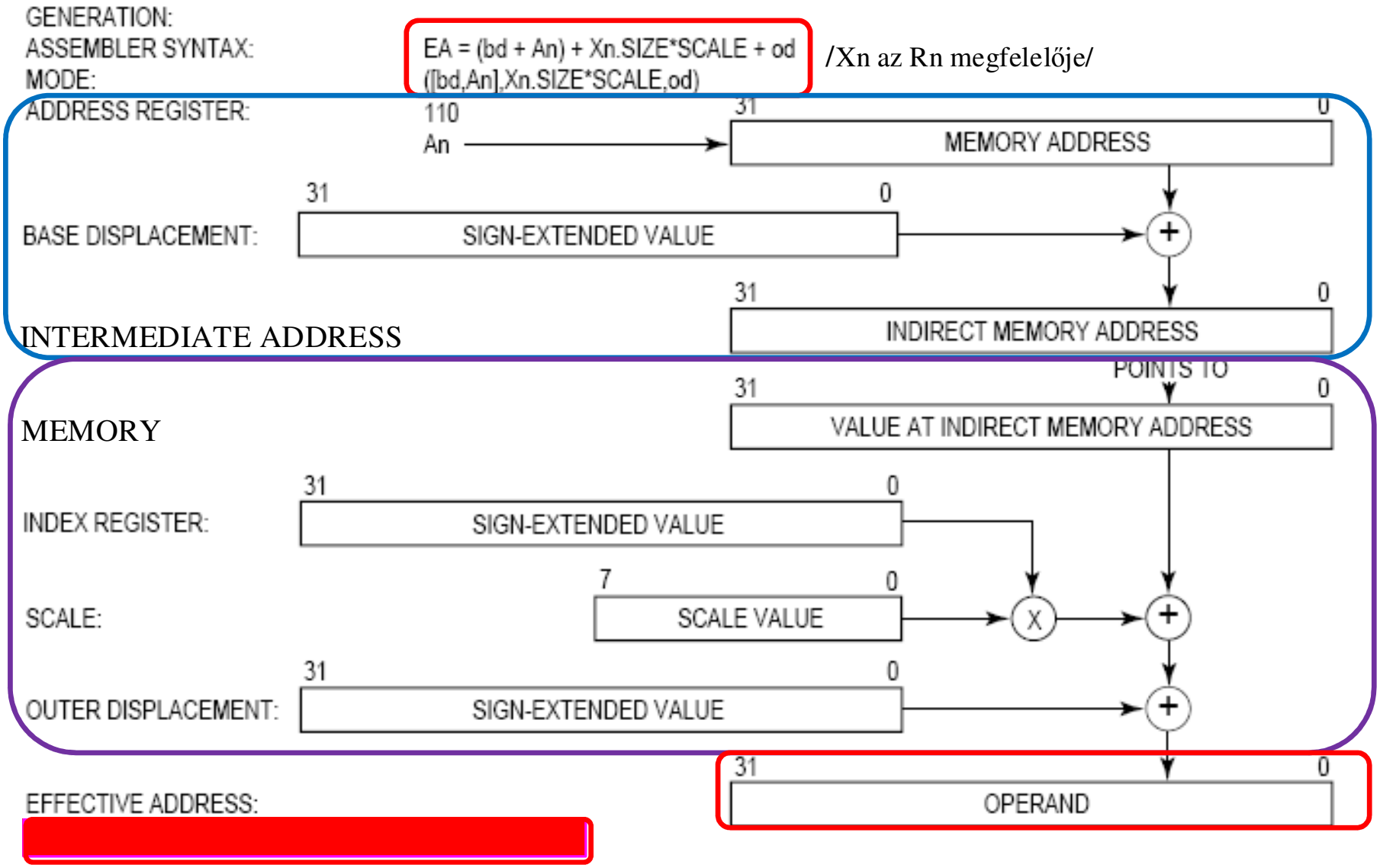
- Többkomponensű címzés (*az előzők kombinációi)*
  - *PC indirekt /relatív/ indexelt címzés eltolással*



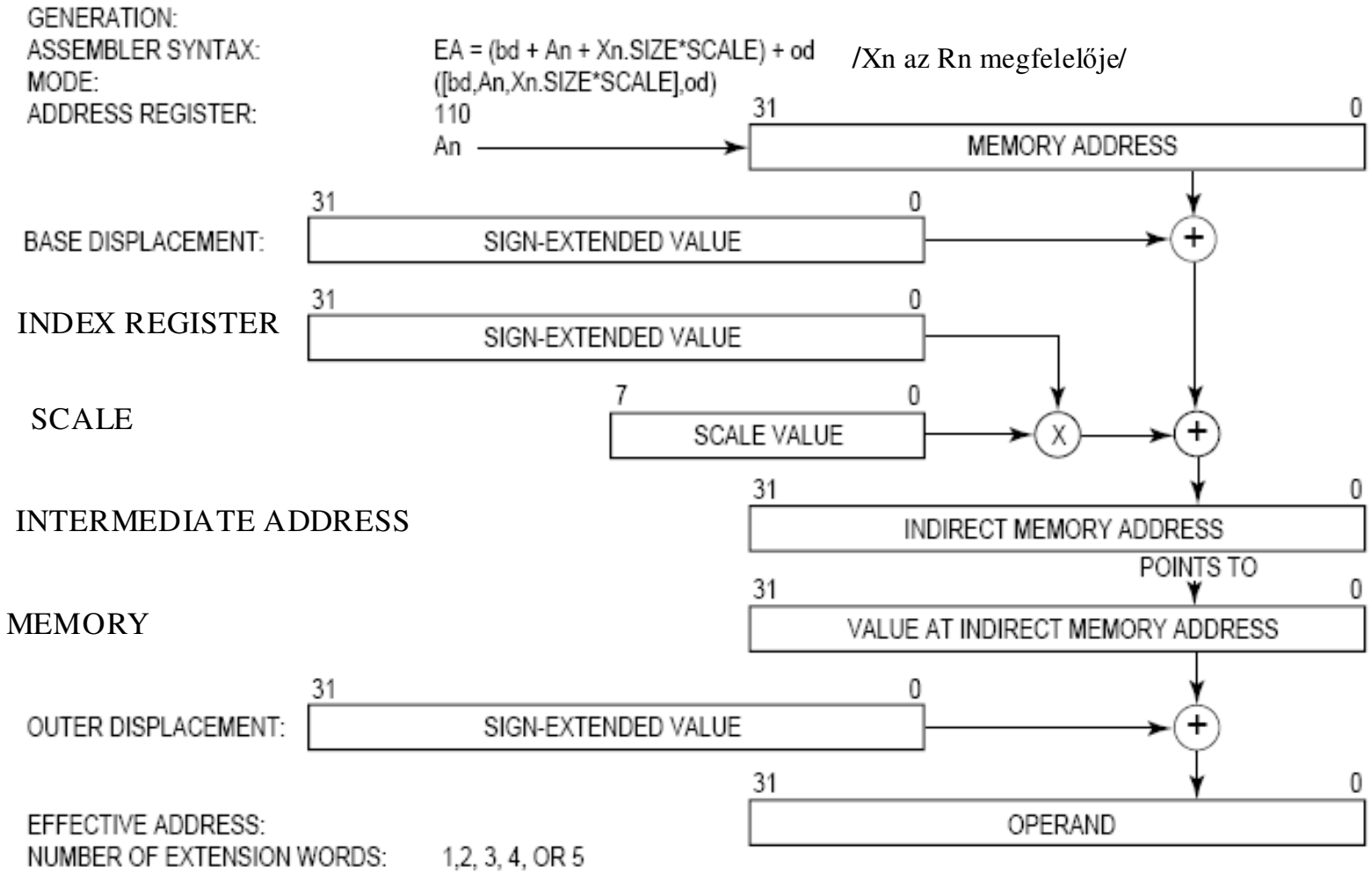
Az immediate címzés is megvalósítható PC relatív címzéssel

/ Xn, és d elnyomással/

Memória indirekt posztindexelt mód



Memória indirekt preindexelt mód



/ An, Xn elnyomásával előállítható az abszolút memória címzés /

❑ ***Bonyolult, időigényes ➔ önálló címaritmetika***

Utasítások kódolása, utasítás formátumok

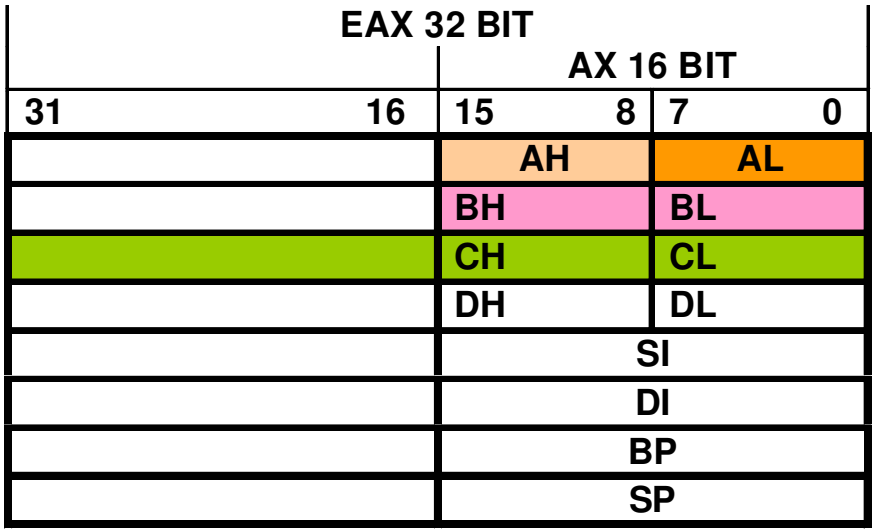
- kódsűrűség növelése
  - ❑ állandó hosszúságú *utasítás* ←→ változó hosszúságú *utasítás*
- szisztematikus kódolás ➔ egyszerűség
- ***Ortogonalitás***
- ***Következetesség***
- ***Kompatibilitás***
- ***Programozhatóság***

- ESETTANULMÁNYOK (Utasítás készlek)

Intel utasításformátum Esettanulmány

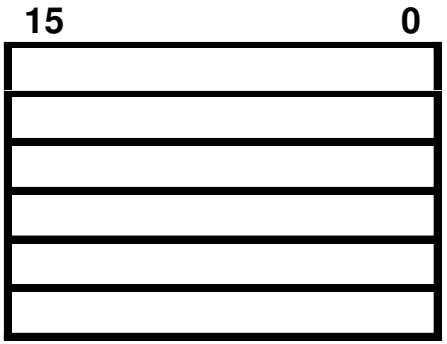
80386 Felhasználói Regiszter modell /ISA IA-32/

GENERAL REGISTERS

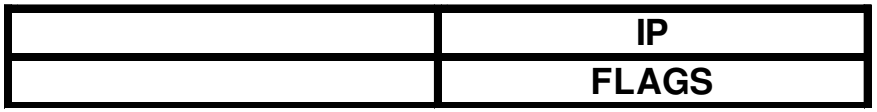


AX	EAX	EAX akkumulátor
BX	EBX	EBX bázisregiszter /pointer/ /DS/
CX	ECX	ECX számláló sztringhez és hurokhoz
DX	EDX	EDX általános I/O cím
SI	ESI	ESI forrás mutató /DS/
DI	EDI	EDI cél mutató /ES/
BP	EBP	EBP bázisregiszter /SS/
SP	ESP	ESP stack pointer /SS/

SEGMENT SELECTOR REGISTERS



CS	Kód
SS	Stack
DS	Adat
ES	Adat
FS	Adat
GS	Adat



IP	EIP	Utasítás számláló
	EFLAGS	Flag regiszter

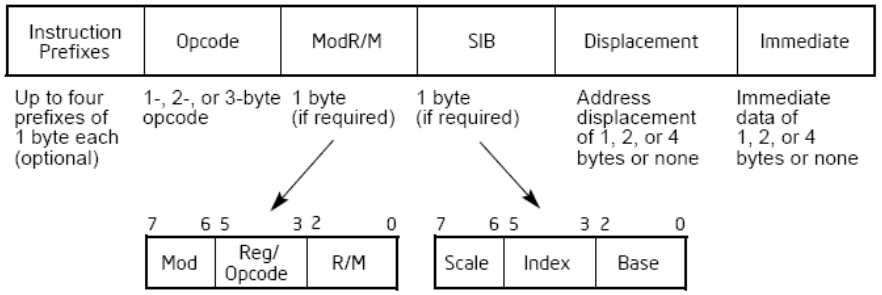
Intel utasításformátum Esettanulmány

INTEL CORE MICRO-architektúra Regiszter modell /ISA Intel-64/

Software-Visible Register or Data Structure	Legacy and Compatibility Modes			64-Bit Mode		
	Name	Number	Size (bits)	Name	Number	Size (bits)
General-Purpose Registers	EAX, EBX, ECX, EDX, EBP, ESI, EDI, ESP	8	32	RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8-15	16	64
Floating-Point Registers[1]	ST0-7	8	80	ST0-7	8	80
Multimedia Extension Registers[1]	MM0-7	8	64	MM0-7	8	64
Streaming SIMD Extension Registers	XMM0-7	8	128	XMM0-15	16	128
Instruction Pointer	EIP	1	32	RIP	1	64
Flags	EFLAGS	1	32	EFLAGS	1	32
Stack Width	—		16 or 32	—		64

Intel utasításformátum Esettanulmány

Intel 64 és IA-32 ISA utasításformátum (1.5 címes)



Prefix: kijelölt kódok

Lock, Rep

/utasítás „zárolása”, ismétlés (pl ***F3 Rep****)/*

Szegmens hozzárendelés felülírása

Operandus méret felülírás

Cím méret felülírás (pl.: 16 - 32 bites cím)

Opkód második byte :

- 0F /escape (menekülő)/ kód után a második byte

- Kötelező előtag után 0Fés2. byte opkód

Pl.: F3 0F E6 /Itt F3 nem jelent REP prefixet!!! Dekódolás!!!/

Opkód harmadik byte

- 0F és két további byte /3byte/

- Kötelező előtag után 0F és még két. byte opkód (pl.:SSE) /4byte/

Pl.: 66 0F 38 01 /Itt 66 (F2, F3) nem jelent op méret felülírás prefixet!!!!/

Néhány utasításnál még Reg/opkód mező 3 bitje is opkód /DEKÓDOLÁS!!!/

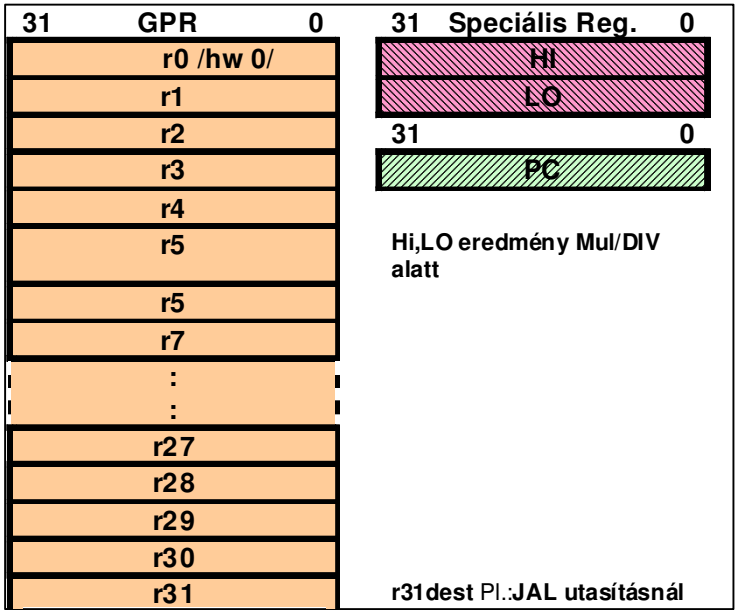
- *Szabálytalan /1-15 byte/ bonyolult, nehezen alkalmazható*
- *Nehéz dekódolni (1-15 byte)*
- *CISC*

MIPS utasításformátum Esettanulmány

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

MIPS-32 Felhasználói Regiszter modell

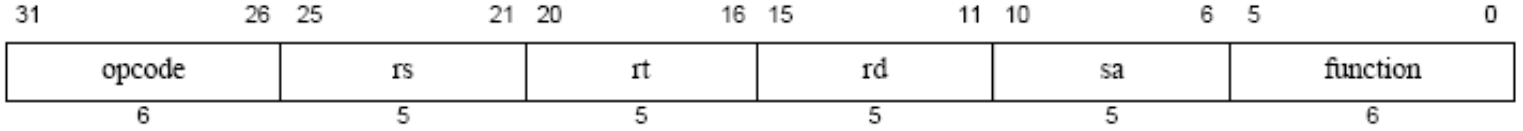
/1981 John L. Hennessy, Stanford University/



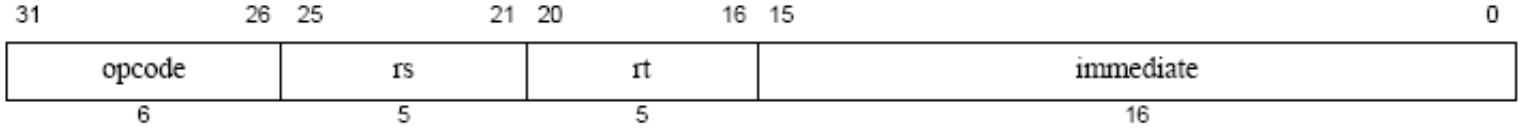
- ***32 32-bites regiszter***
- ***Korlátozások az alkalmazásukban***
- ***Azonos utasítás méret (32 bit)***

MIPS utasításformátum Esettanulmány

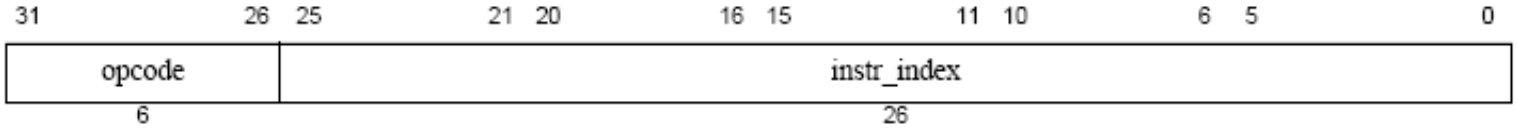
Regiszter típusú



Immediate típusú



Jump típusú



6 bites opkód !!! / + function/

rs, rt forrásregiszterek, rd célregiszter /háromcímes/

sa shift /léptetés/ száma,

add rd,rs,rt kód: [0 rs rt rd 0 0x20] (rd=rs+rt)

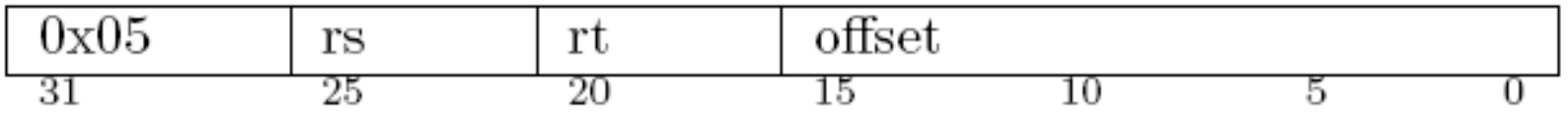
sra rd, rt, sa [0 0 rt rd sa 0x03] shift jobbra „sa" lépést

addi rt,rs, imm [0 rs rt imm] (rt>>rd !!!!) (rt=rs+imm)

Pszeudo utasítás pl.:mov rd,rs add rd,r0,rs

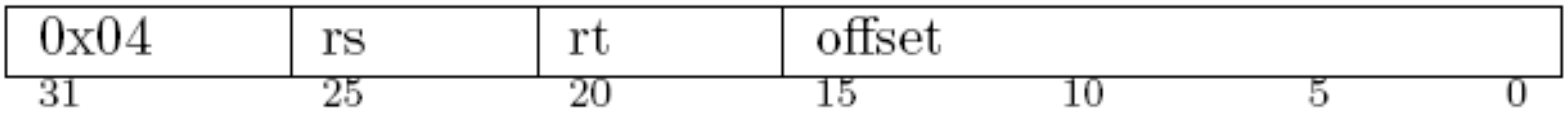
kód: [0 rs $zero rd 0 0x20]

bne rs, rt,offset



ugrik offset értéket ha rs nem egyenlő rt

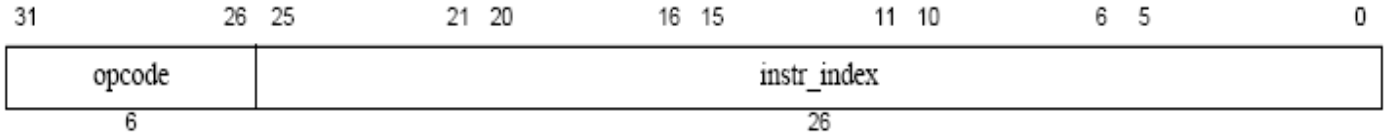
beq rs, rt,offset



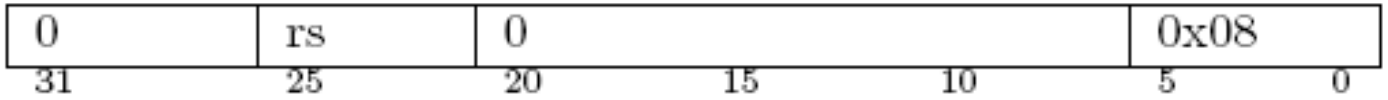
ugrik offset értéket ha rs=rt

j target ***PC-ben átírja a 2-27biteket /target 0-25 bittel/+00***

Jump típusú



jr rs

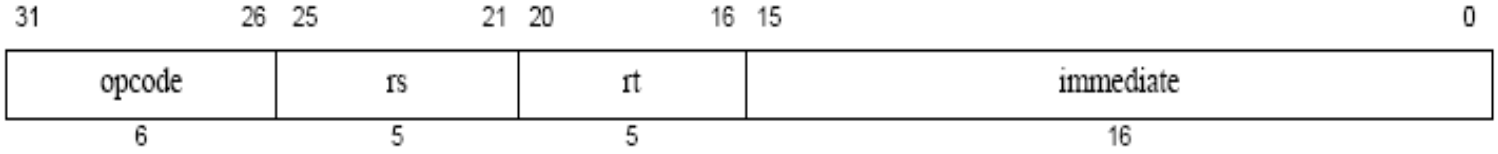


Ugrik (rs) címre

Jal ***jump and link r31:=PC+4, PC<<target***

MIPS utasításformátum Esettanulmány

Immediate típusú



lw rd, offset (rs) *load 32 bites szó rd-be offset+(rs) címről*

/itt rt tölti be rd szerepét/

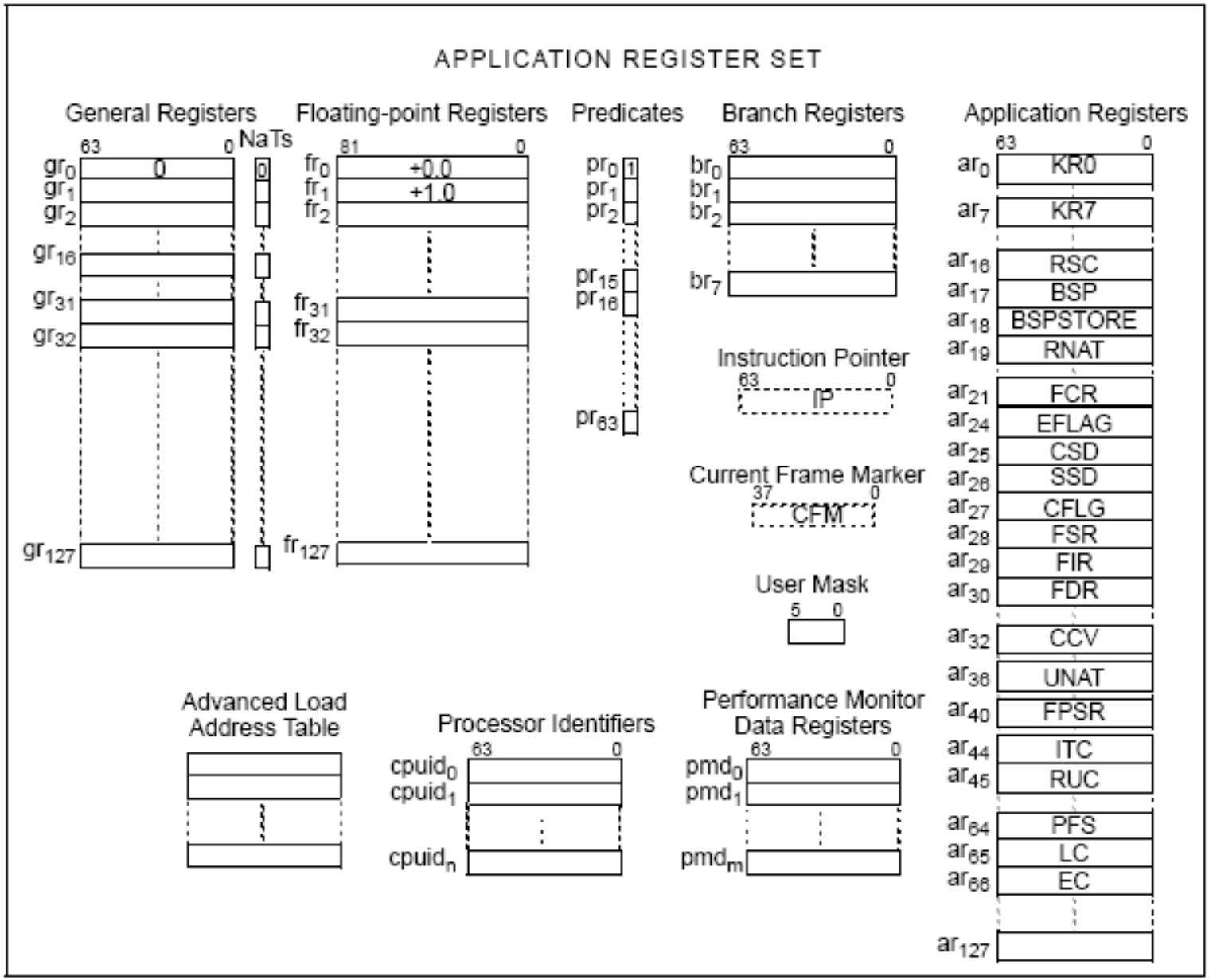
lw r1,20(r3) >>r1=M(r3+20), *lw r1,432(r0)>>r1=M(432) /r0==0/*

sw rt offset (rs) *tárolja rt tartalmát offset+(rs) címre*

sw r2, 0(r1) >>M(r1)=r2 *swr 2,23(r0)*

- *Szabályos 32 bites RISC utasítások*
- *Könnyen megvalósítható*
- *könnyen alkalmazható /lsd. regiszterhasználat/*
- *Nagyon egyszerű címzés /gyors regiszter elérés/*
- *Memóriához csak load/store /ez az ára a sebességnek/*
- *Fentiek miatt előnyösen alkalmazható a részműveletek szalagrendszerű feldolgozása* ***/lsd. később pipe-line/***

Intel Itanium Architektúra / MIMD ISA modell/



Intel Itanium Architektúra / MIMD ISA modell/

VLIW */Very Long Instruction Word/*

EPIC */Explicitly Parallel Instruction Computing/*

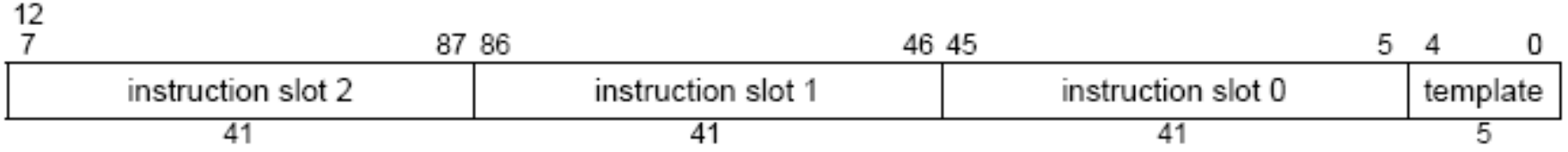
Utasítás szintű párhuzamosság /ILP *(Instruction Level Parallelism)* /

➔ IPC nő

Alapvetően Load/Store architektúra, RISC típusú utasítások

Instruction Type	Description	Execution Unit Type
A	Integer ALU	I-unit or M-unit
I	Non-ALU integer	I-unit
M	Memory	M-unit
F	Floating-point	F-unit
B	Branch	B-unit
L+X	Extended	I-unit/B-unit

Három utasításból álló 128 bites utasításköteg /a fordító készíti/



Intel Itanium Architektúra /ISA MIMD modell/

A Template /sablon/ a párhuzamosan végrehajtható műveletekre

Template	Slot 0	Slot 1	Slot 2
00	M-unit	I-unit	I-unit
01	M-unit	I-unit	I-unit
02	M-unit	I-unit	I-unit
03	M-unit	I-unit	I-unit
04	M-unit	L-unit	X-unit[a]
05	M-unit	L-unit	X-unit[a]
06			
07			
08	M-unit	M-unit	I-unit
09	M-unit	M-unit	I-unit
0A	M-unit	M-unit	I-unit

Jelzi, a valamelyik utasítást követő egy vagy több utasítás - esetleges erőforrás - függőséget (meg kell állítani) /pl.: 03 slot1 után/

Több köteg összeláncolható
Predikáció (feltételes utasítás végrehajtás az elágazások helyett)
Fordítási időben végzi az átalakítást

- **RISC utasítások**
- **Nagymértékben párhuzamosított**
- **IA-32-t is tud**

Utasítás készlet tulajdonságai

Az utasításkészlet két fő típusa		
CISC	⟷	**RISC**
sok *100-500* komplex	**- utasítás -**	**kevés *50-100* egyszerű**
sok *10-20* bonyolult	**- címzési mód -**	**kevés *2-4* egyszerű**
kevés *8-32*	**regiszter**	**sok /32-…x100/**
regiszterben és memóriában is	**- operandus -**	**regiszterben memória elérésre csak Load/Store**
változó-hosszú több órajel idejű	**műveleti idő**	**állandó-rövid egy órajel idejű is**

- CISC tervezési szempontjai
  - *Ortogonalitás*
  - *Következetesség*
  - *Kompatibilitás*
  - *Programozhatóság /**Kinek a szempontjából?**/,*
- *Magas szintű nyelvek támogatása*
- *Operációs rendszerek támogatása*

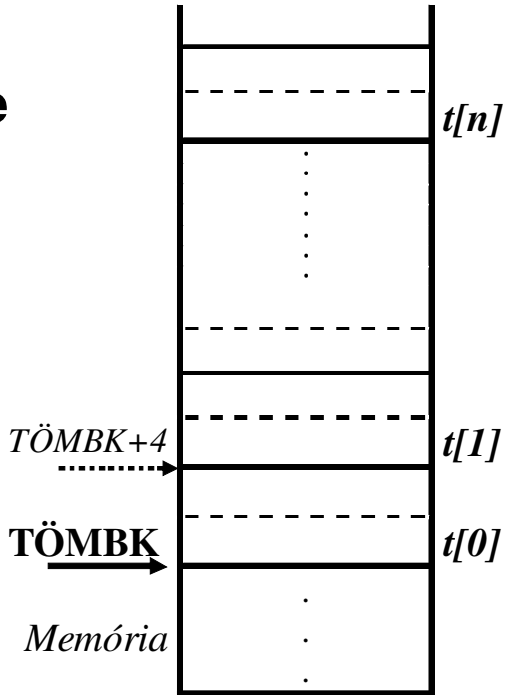
Magas szintű nyelvek támogatása

- Speciális adattípusok támogatása
  /BCD, lebegőpontos, karakterlánc, stb./
- Összetett adatszerkezetek kezelése
  - **Tömbök indexelt címzés**
  - **Rekordok** **kétkomponensű címzés, indirekt címzés**
  - **Láncolt listák** **speciális utasítások**
    pl.: mutatók kezelése

$EA_{t[n]} = TÖMBK + n \cdot S$

Írjuk be t[0]-ba a 2016, t[1]-be 10 értéket

```
LEA (TOMBK.L),A1          ;kezdő cím A1-ben
MOVE.L #2016,(A1)+  ;2016 a t[0]-ban
MOVE.L #10,(A1)+       ;10 a t[1]-ben
Nagyobb adat méret esetén indexregiszter!!
Pl.:MOVE.L #xx,(TOMBK,D1.L*s)   ; D1 index
```



Rekord

```
type hallgato = RECORD
név : ARRAY[1..20] OF CHAR;
ID :  INTEGER;
atl : REAL;
END;
hallg:ARRAY[1…400] OF hallgato;
hallg[10].atl:=4.95;
```

EA:= hallg + atl offset;

Move (offset, hallgató) ← 4,95

Vagy EA := tömbkezdő + (ind * méret) + offset

Pl.:

```
MOVE.W  #10,D1
MULU.W  #26,D1                  ; hallg[10]
LEA     (hallg.L),A1
MOVE.L  #4.95,(22,A1,D1.W)  ; hallg[10].atl
```

Struktúra

```
typedef struct
{
char nev [20];
short int ID;
Float atl;
} hallgato;

hallgató hallg[400];
        .
hallg[9].atl=4.95;
```

- **Mutató, láncolt lista**

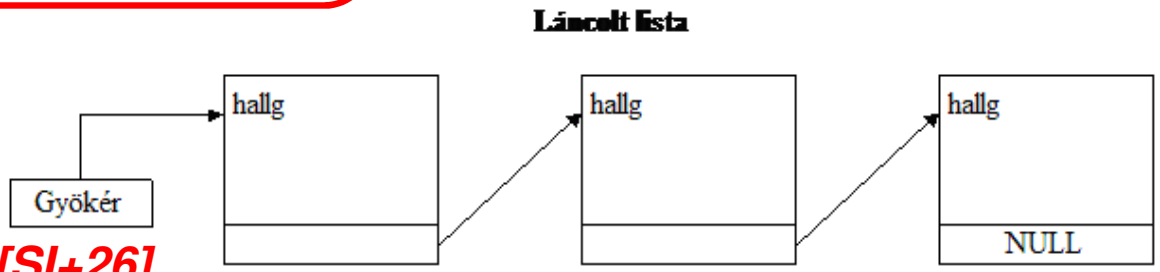
  speciális utasításokkal gyökér

Pl.: Intel

hallgp → ES : SI + 26 → SI

ES : SI + 28 → ES váltás?

Külön utasítás: LES ***LES SI, ES:[SI+26]***



Operációs rendszer funkciók támogatása

- Rendszerhívások
- Védelmi rendszer
- Párhuzamos futtatás támogatása
- Programszerkezetek támogatása
  - Eljárások, függvények
    - Szubrutinhívás + verem használat
    - Speciális utasítások ***/pl.: ENTER, LEAVE/***
  - Programmodulok kezelése
    - Bázis relatív címzés

Eljárások futtatási modellje

- Magas szintű nyelv eljárásainak megvalósítása:
  - Globális változók *fix tárcímen* ***(mindig látszanak)***
  - Paraméterek
    - Fix tárcímen*?* → *nem lehetne rekurzív*
    - Regiszterben? → *kevés van*
    - Paraméterlista *(mutatók átadása)* →*bonyolult*
    - Veremben →*OK*
    - *Verem keret (STACK FRAME)*
      → Egységes kezelés
    - Lokális változók ~ paraméterek (egységes kezelés)
    - Visszatérési érték → regiszterben

Verem keret *(stack frame)*

1. Paraméterek veremre írása
2. Szubrutin hívás
3. Implicit paraméterek /esetleges regiszterek/, hívó keretcímének mentése a verembe
4. Hívott keretének beállítása
   >>> ehhez képest címezzük meg az eljárás objektumait
5. Helyfoglalás lokális változóknak
6. Eljáráson belül hivatkozás lokális változókra, paraméterekre
7. Visszatérési érték beállítása
8. Lokális változók helyének felszabadítása
9. Hívó keretének visszaállítása
10. Visszatérés a hívóhoz
11. Paraméterek lebontása

Példa: ”PASCAL”

FUNCTION f (i, j : INTEGER):INTEGER;

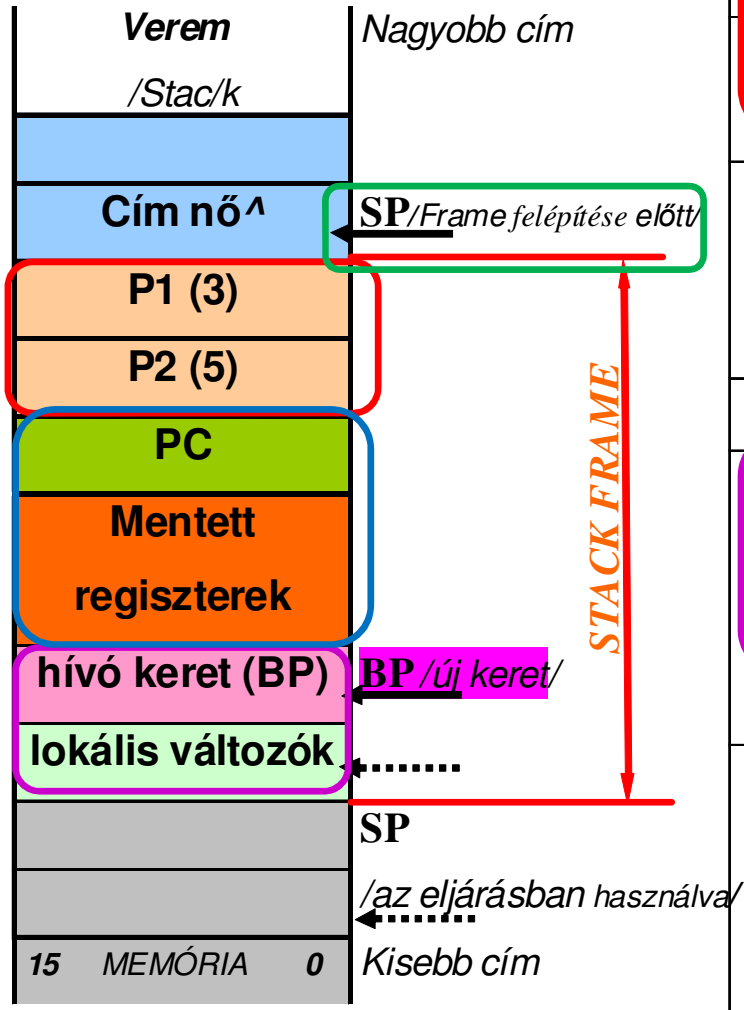
VAR I : INTEGER;

BEGIN ... END;

Hívás pl.: f(3, 5)

”C”

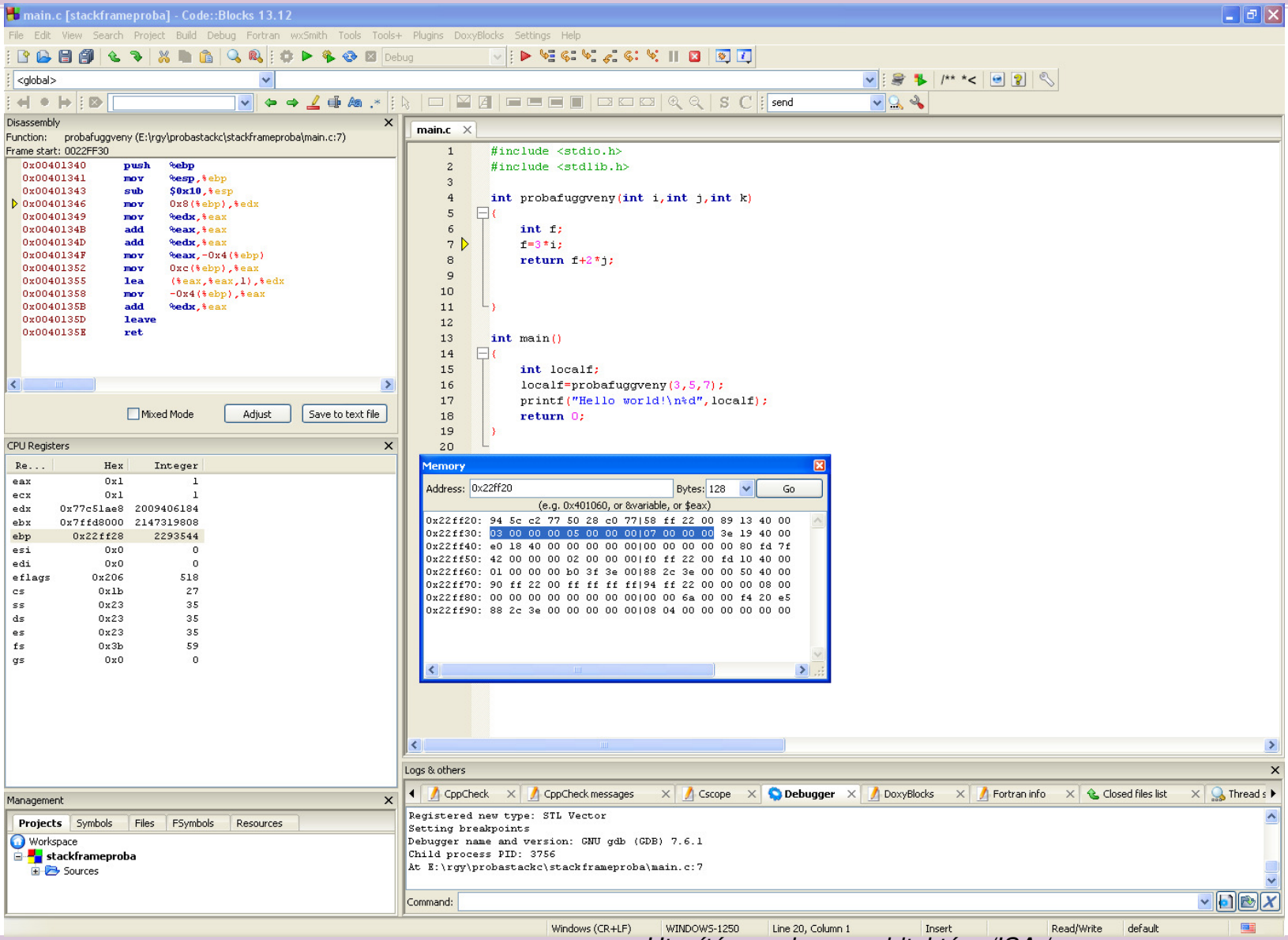
```
int f(int i, int j)
{ int I;
  .
  .
}
```



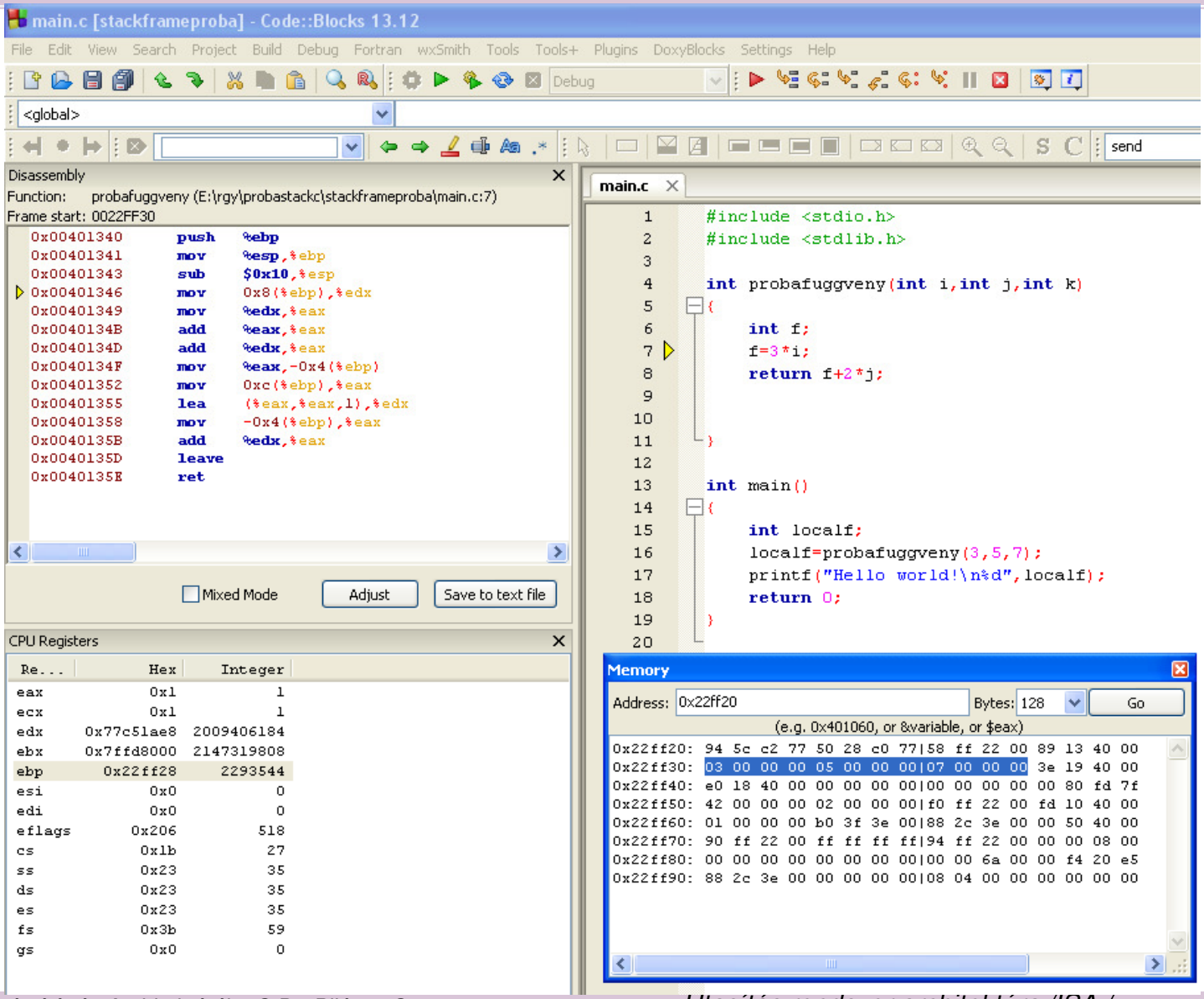
Verem keret /stack frame/

	80486	8086	„C”
1	PUSH 3	MOV AX,3 ; P1	„P2”
		PUSH AX	
1	PUSH 5	MOV AX,5 ; P2	„P1”
		PUSH AX	
2	CALL f	CALL f	
	...	...	; ADD SP,4 11
	...	...	
3	f:	f:	
3	ENTER 2,0	PUSH BP	
4		MOV BP,SP	
5		SUB SP,2	
	.....	.....	
6	MOV AX, WORD PTR [BP-2]	MOV AX, WORD PTR [BP-2]	
7	MOV AX, ..	MOV AX, ..	
8	LEAVE	MOV SP, BP	
9		POP BP	
10	RET 410,11	RET 410,11	; RET 10

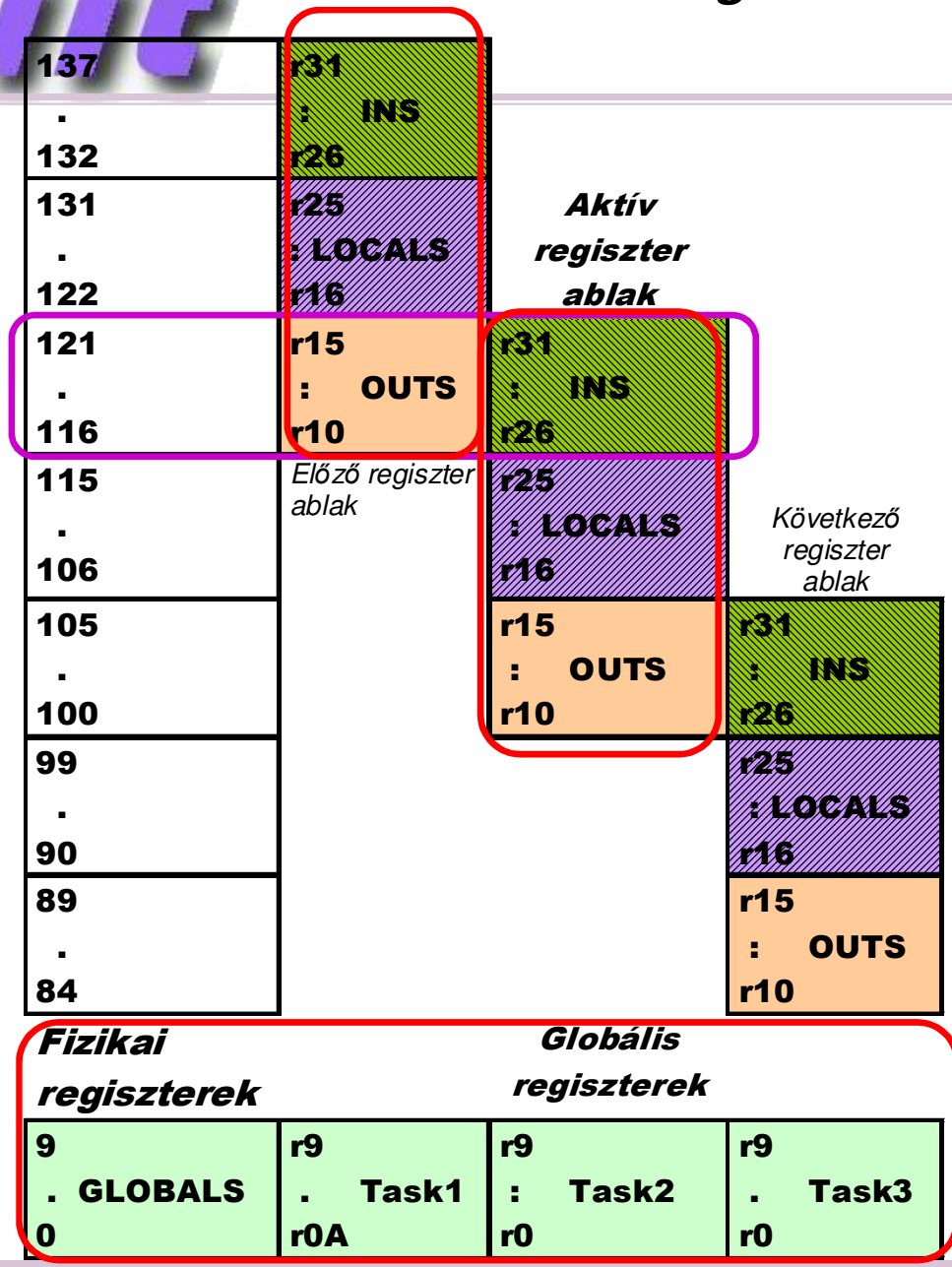
Stack frame-C



Stack frame-C

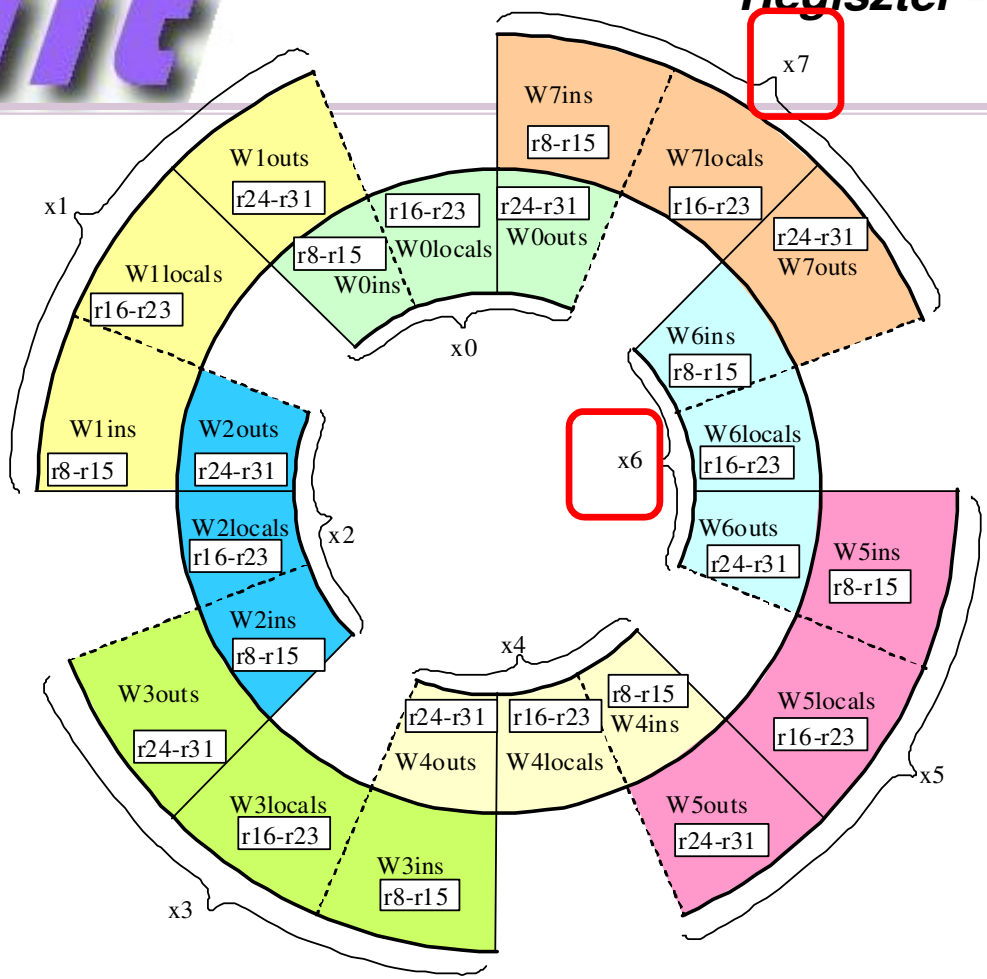


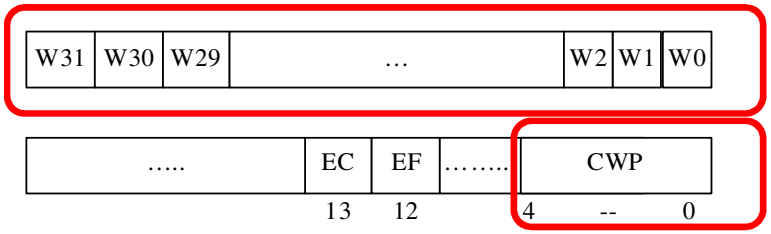
RISC II: *138 regiszter /ablaktechnika 10+ 8W x(6+10+6) /*



- A regisztereket csoportokba /ablakok/ szervezik
- Egy pointer jelöli ki a regiszterblokkban az aktív ablakot, azon belül logikailag címez
- A globális regiszterek mindig aktívak
- A bemenő adatok és a kimenő adatok regiszterei a szomszédos csoportokban átfedik egymást
- Gyors kontext váltás

Regiszter - ablaktechnika





- CWP aktív ablakmutató
  /eggyel csökkentve a következő ablakot aktiválja, eggyel növelve pedig az előzőt/
- Minden ablakhoz egy tiltás bit (W0-W31)
  - /a túlcsordulás elkerülésére, és egyes csoportok elszeparálására/
  - /az ábrán x7 az első, x0 az utolsó/

Stack frame

Egy ***Pascal*** programban adott a következő ***függvény***:

```
function f(i;j:integer):integer;
  var m:integer;
begin …….
end;
```

(az integer 16 bites)

8086-os processzornál a fenti függvényt meghívtuk az aktuális paraméterekkel. A mellékelt ábrán a memóriának az a része látható amelybe a verem keret használatakor szokásos szerkezet (stack frame) is felépült. Az SP a keret felépítése után a végrehajtás alatt a bejelölt helyre mutat. Adja meg, hogy az ***i*** paraméter milyen aktuális értékével hívták meg a függvényt.

i=1200. H

Mi az ***m*** lokális változó pillanatnyi értéke? ***m=0A00*** H

Mi a stackpointer értéke közvetlenül a visszatérés után? ***SS:SP=:EFA02***H

C nyelvű függvény esetén mi lenne az ***i*** (első paraméter értéke? ***i=0B12***.H

Mem.cím(hexa)	
EFA02	0011H
EFA00	1200H
EF9FE	0B12H
EF9FC	CS
EF9FA	IP
EF9F8	AX
EF9F6	SI
EF9F4	BP
EF9F2	0A00H
SS:SP →EF9F0 végrehajtás alatt ere bővül	0A01H
EF9EE	1221H
EF9EC	0012H
EF9EA	0BC2H

Összefoglalás

- Utasítás architektúra
- Utasítások felépítése, kialakítási szempontjai
- Regiszter modell
- Műveletek csoportosítása
- Operandusok címzése
- RISC↔CISC sajátosságok
- Magas szintű nyelvek támogatása
- Veremkeret (stack frame)