2. gaia

Agindu multzoa

Konputagailuen Egitura

Makina lengoaia eta mihiztadura lengoaia

- **Makina lengoaia:** kode bitarra, prozesadoreak ulertzen duen bakarra
- **Mihiztadura lengoaia:** makina lengoaiaren adierazpide sinbolikoa

Makina Lengoia	*Mihiztadura lengoaia*	
`11100101100111110001000000001100`	ldr	r1, ald
`11100001101000000010000000000001`	mov	r2, r1
`11100000100000010011000000000010`	add	r3,r1,r2
`11100101100011110011000000000100`	str	r3, batu

Programa mihiztatzailea: mihiztadura lengoaiatik makina lengoaiara Itzultzen du kodea

Agindu multzoaren diseinu filosofiak

- **CISC** (*Complex Instruction Set Computer*)
  - Helburu orokorreko + helburu bereziko aginduak
    - Helburua programatzaileari erraztasunak eskaintzea da
  - Agindu konplexuak: formatu aldakorra, helbideratze modu asko …
  - Agindu 1 → hainbat ekintza
  - Erregistro gutxi: eragiketak zuzenean memoriako datuekin
  - hardware konplexua: exekuzio “motelagoa” RISC makinetan baino

- **RISC** (*Reduced Instruction Set Computer*)
  - Agindu multzo txikia (helburu orokorrekoak)
    - Helburua makinaren errendimendua da
  - Aginduen erabileraren neurri estatistikoetan oinarrituta
  - Agindu sinpleak: formatu finkoa, hebideratze modu gutxi…
  - Erregistro asko: RRR motako aginduak (Load/Store memoira atzitzeko)
  - Hardware sinplea: agindu 1 → ziklo 1 **(segmentazioa)**

- **CISC agindu bat → hainbat RISC agindu**

ARM Arkitektura

Ezaugarri orokorrak:

Prozesadorea	32 biteko RISC prozesadorea
Memoria	8 biteko memoria posizioak (helbideratze unitatea bytea) 32 biteko memoriako helbideak *[0..2^{32}-1]*
Erregistroak	32 biteko 16 erregistro orokor (R0-R12.- erabilera orokorrerako) R15 erregistroa: ***PC*** CPSR erregistroa: 4 adierazle (flag) dauzka (**N**egativo, **Z**ero, **C**arry, eta o**V**erflow)
Datu motak	*Datuak 8 bitekoak (byte), 16 bitekoak (half-word) edo 32 bitekoak (word) izan daitezke.* Zenbaki osoak adierazteko birako osagarria erabiltzen da.
Aginduen formatua	Luzera finkoko aginduak (32 bit)

Programa baten egitura ARM mihiztadura lengoaian

@ oharrak	@ adibidea
.code 32 .global main, __cpsr_mask	.code 32 .global main, __cpsr_mask
. programako aldagaiak . .	X: .word 15 Y: .word 10 batu: .word 0

main:
.
programa nagusiko aginduak
.
.
mov pc,lr
	main:
	ldr r2, X
	ldr r3, Y
	add r5,r2,r3
	str r5, batu
	
	mov pc,lr @ programaren bukaera

. . azpirrutinak . .	

ARM-aren aginduen laburpena

Memoriako atzipena	LDR	*Rh, memoriako helbidea*
	STR	*Ri, memoriako helbidea*
Aritmetikoak	ADD, SUB, MUL	*Rh, Ri, er2*
	ADDS, SUBS, MULS	*Rh, Ri, er2*
Logikoak	AND, EOR, ORR	*Rh, Ri, er2*
	ANDS, EORS, ORRS	*Rh, Ri, er2*
Datu mugimenduak	MOV	Rh, er2
	MOVN	Rh, er2
Desplazamenduak	LSL	*Rh, Ri, er2*
	ASR	*Rh, Ri, er2*
	LSR	*Rh, Ri, er2*
Konparaketak	CMP	*Rn, er2*

Jauzi aginduak

Baldintza gabeko jauziak:

b **etiketa** **(PC PC + despl)**

Baldintzapeko jauziak:

✓ *Berdin:*	**be**	**etiketa**	(**flag** Z = 1)
✓ *Desberdin:*	**bne**	**etiketa**	(**flag** Z = 0)
✓ *Txikiagoa:*	**blt**	**etiketa**	(**flag** N = 1)
✓ *Txikiagoa edo berdina:*	**ble**	**etiketa**	(**flag** N = 1 edo Z=1)
✓ *Handiagoa:*	**bgt**	**etiketa**	(**flag** N = 0 eta Z=0)
✓ *Handiagoa edo berdina:*	**bge**	**etiketa**	(**flag** N = 0 edo Z=1)

if ...else egitura

```
#include <stdio.h>

main()
  {
            if (baldintza)
               {
                     /*baldintza betetzen bada*/
               }

             else
               {
                      /* baldintza ez bada betetzen */
               }
  }
```

(baldintza) ? true : false;

if ...else egitura

`handi` aldagaian a eta b bi aldagairen arteko handiena gordetzen duen programa.

```
.code 32
.global  main, __cpsr_mask

a:              .word 15
b:              .word 16
handi:          .word 0

main:
          ldr   r1, X
          ldr   r2, Y
          cmp r1, r2
          ble  else
          str   r1, handi
          b     buk
  else:   str  r2, handi
  buk:      mov  pc, lr
```

```
main()
{
int a=15, b=16, handi;
        if (a > b) handi = a;
       else handi = b;
// handi = (a > b) ? a : b;
}
```

Bektoreak

- Bektoreak atzitzeko helbideratze modua:
  oinarri-erregistro + indize-erregistro ***[rb + rx]***
- Erregistro batean oinarri-helbidea (bektorearen hasierako helbidea) kargatzeko:
  - ***ldr r1,=B1*** | ***adr r1,B1***

```
B1:  .word 7,8,-9
B2:  .word 0,0
```

...	...	...	...		
00	00	00	07	@100	B1:
00	00	00	08	@104	
FF	FF	FF	F6	@108	
00	00	00	00	@112	B2:
00	00	00	00	@116	

```
adr   r1,B1          @  r1 = @B1
mov   r2,#1
lsl   r3,r2,#2
ldr   r3,[r1,r3]     @r3:=B1[1]
```

@mem=@(edukia(r1)+edukia(r3)) =>@104

r1= hasierako helbidea ***r3= i*l***

i.- osagaiaren indizea

l .- osagai bakoitzaren luzera bytetan

Bektoreak

Bektore baten bi osagairen batura kalkulatzen duen programa

```
.code 32
.global main, __cpsr_mask

bek :       .word   -1,4,6,-9,78,12,-34,0,-1,612
i:          .word 4
batura:     .word 0

main:
    adr     r1, bek
    ldr     r2, i
    lsl     r4,r2,#2
    ldr     r3,[r1,r4]
    add     r2,r2, #1
    lsl     r4,r2,#2
    ldr     r5,[r1,r4]
    add     r3, r3,r5
    str     r3, batura
    mov  pc,lr
```

```
main()
{
 int i=4, batura=0;
 int bek[10]={-1,4,6,-9,78,12,-34,0,-1,612};

batura=bek[i]+bek[i+1];
  }
```

for egitura

```
#include <stdio.h>

main()
{
        int i, n, zen,kont,batu=0;

        printf(“Esan batu beharreko zenbaki kopurua:\n");
        scanf("%d", &kont);
        for (i=0;i<kont; i++ )
          {
            printf(“Sartu zenbakia:\n");
            scanf("%d", &zen);
            batu+=zen;
          }
        printf("%d",zen);
}
```

Hasieratze eta eguneraketa espresioak konposatuak izan daitezke:

for (x=0, y=10; x<y; x++, y--)

for egitura

Bek bektoreko osagai guztien batura kalkulatzen duen programa

```
.code 32
.global main, __cpsr_mask

Bek:        .word 3, 2, 3, -5, 4, -3, 6, 5, 31, -6
batu:       .word 0

main:
mov   r0, #0
adr     r1, Bek
mov   r2, #0
FOR:        cmp   r2, #10
beq    buk
lsl      r4,r2,#2              @ezk. Desplazatu - bider 4
ldr      r3,[r1,r4]
add    r0, r0,r3
add    r2, r2, #1
b        FOR
buk:        str      r0, batu
mov   pc,lr
```

```
main()
{
 int i, n=10,batu=0;
 int Bek[10]={3,2,3,-5,4,-3,6,5,31,-6};

       for (i=0;i<n;i++)
            batu=batu+Bek[i];
  }
```

for egitura

B1 eta B2 bi bektoreren batura kalkulatzen duen programa. Emaitza BATURA bektorean gordetzen da.

```
 .code 32
 .global      main, __cpsr_mask
B1:           .word 3, 2, 3, -5, 4, -3, 6, 12, 1, -9
B2:           .word 7, 6, 9, 15, -5, -11, 2, 5, 31, -6
BATURA:   .word 0,0,0,0,0,0,0,0,0,0

 main:
        mov   r0, #0        @ r0= indizea
        adr     r1, B1
        adr     r2, B2
        adr     r3, BATURA
FOR: cmp   r0, #10
        beq   buk
        lsl     r4,r0,#2
        ldr     r5,[r1,r4]
        ldr     r6,[r2,r4]
        add   r5, r5,r6
        str     r5,[r3,r4]
        add  r0, r0, #1
        b  FOR
buk:  mov    pc,lr
```

```
main()
{
 int i, n=10, BATURA[10];
 int B1[10]={3, 2, 3, -5, 4, -3, 6,12,1, -9};
 int B2[10]={7,6,9,15,-5, -11,2 5,31, -6};
for (i=0;i<n;i++)
            BATURA[i]=B1[i]+B2[i];
}
```

WHILE egitura

X eta Yren zatitzaile komunetako handiena kalkulatzen duen programa (X>0 eta Y>0):

```
main()
{
int X, Y,zkh;
printf("Sartu X eta Y-ren balioak:\n");
scanf("%d %d", &X, &Y);
while (X!=Y)
      if (X>Y) X- =Y;
                  else Y- = X;
zkh=Y;
printf("Zatitzailea komunetako handiena: %d", zkh);
}
```

```
.code 32
.global     main, __cpsr_mask
 main:
ldr r1, x
ldr r2, y
while: cmp r1, r2
beq buk
blt txikia
sub r1, r1, r2
b while
txikia: sub r2, r2, r1
b while
buk: str r1, zkh
       mov pc, lr
x: .word 12
Y: .word 18
zkh: .word 0
```

do...while(REPEAT) egitura

`zenb` aldagai baten kodeketan 1 balioa hartzen duen bit kopurua kontatu eta `batekoak` aldagaian gorde behar da.

```
. code 32
.global     main, __cpsr_mask
 main:
ldr      r1, zenb
mov   r2, #0
mov   r3,#32
repeat:     and    r4, r1 ,#0x80000000
cmp    r4,#0
beq    zero
add    r2,r2,#1
zero:       lsl      r1, r1, #1
            subs  r3, r3, #1
bne    repeat
str      r2, batekoak
mov pc, lr
zenb: .word -27
batekoak: .word 0
```

```
main()
{  int batekoak=0, bitak=32;
   int ema, zenb=40;
do {
      ema=zenb & 1;
      if (ema!=0) batekoak++;
      zenb>>=1;
      bitak- -;
      } while (bitak>0);
}
```

little-endian eta big-endian adierazpideak

Byte bat baino handiagoak diren datuak memorian gordetzeko bi aukera:

- ***Little-endian*** datuaren pisu txikieneko bytea memoriako helbide txikienean gordetzen da
- ***Big-endian*** datuaren pisu txikieneko bytea memoriako helbide handienean gordetzen da

Adibidea: 00 03 A1 0F (4 byte hamaseitarrez) datua emanda

Memoria-helbidea	**@i**	**@i+1**	**@i+2**	**@i+3**
Little-endian	0F	A1	03	00
Big-endian	00	03	A1	0F

Byte barruko biten ordena ez da aldatzen

Datuak memorian

- Datu tamaina desberdinak: ***.byte***, ***.hword eta .word***
- Datuak memorian lerrokatuta gorde behar dira:
  - ***hword*** motako datuak 2-ren multiplo diren helbideetatik hasita gorde
    .align 1 -> *2 byterako lerrokatzea*
  - ***word*** *motako datuak 4-ren multiplo diren helbideetatik hasita gorde*
    .align -> *4 byterako lerrokatzea*

…	…	…	…	
00	04	03	0F	@2C0
--	--	00	05	@2C4
00	00	00	07	@2C8
00	02	00	09	@2CC
E5	1F	00	00	@2D0

```
d1:     .byte 15,3
d2:     .hword 4
d3:     .hword 5
        .align
d4:     .word 7
d5:     .hword 9
d6:     .word 2
```

- Ez da onartzen, ez dago lerrokatuta -> .align jarri aurretik

Aginduen exekuzio faseak

Aginduen exekuzioa ondoko faseetan banatzen da.

Agindu guztiak ez dira fase guztietatik pasatzen.

B	D	Ir	A	M	Id
PC, M, IR	IR	EM	UAL	Memoria	EM
IR := M[PC] PC := PC + 4	deskodetu (EK)	Irakurri ri (EM)	Eragiketa	MEM irakurri edo idatzi	Idatzi ri (EM)

Mihiztatzea, estekatzea eta karga

Konpiladorea:

Goi-mailako lengoaia ⇒ Makina lengoaia

Mihiztatzailea:

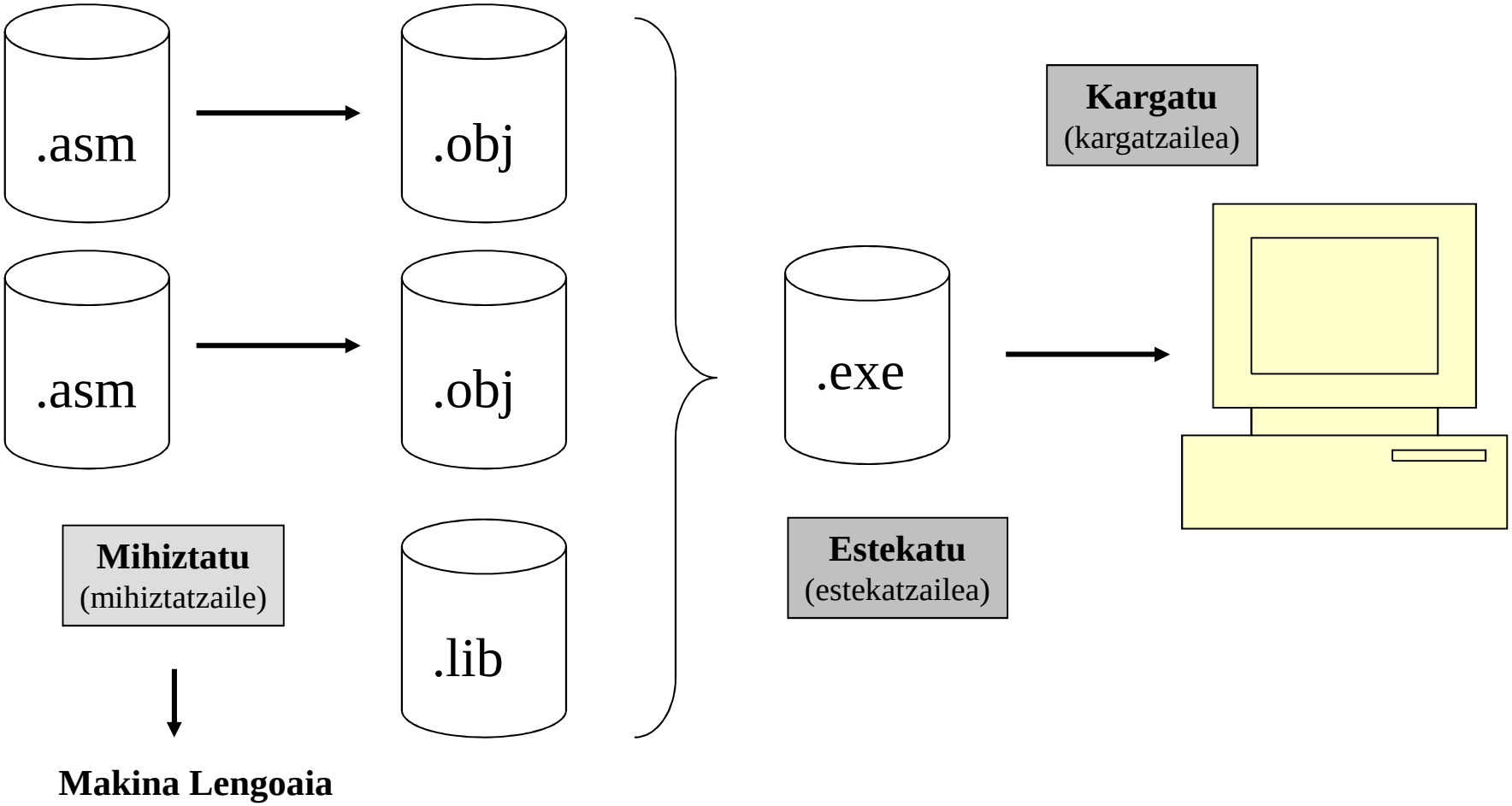
Mihiztadura lengoaia ⇒ Makina lengoaia

n objektu modulu → modulu exekutagarri 1

kanpo erreferentziak ebatzi

erabilpena: liburutegiak, konpilazio banatua

Mihiztatzea, estekatzea eta karga

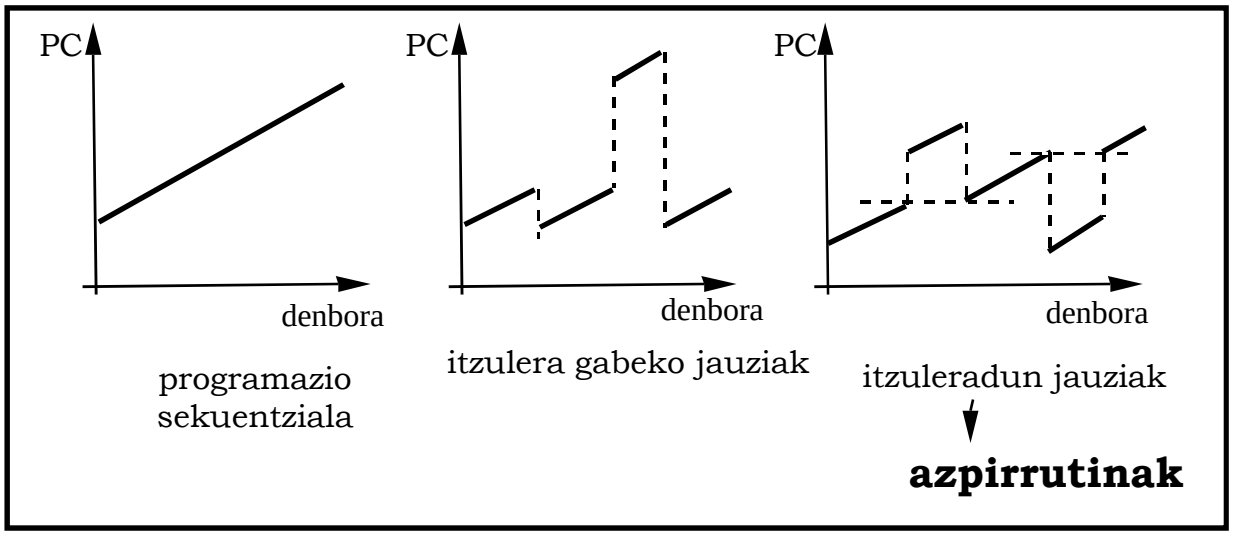


2.gaia.- Agindu multzoa

1. Makina lengoaia eta mihiztadura lengoaia
2. Azpirrutinak
3. Aginduen formatua
   - Helbideratze moduak
4. Makinen sailkapena aginduen formatuaren arabera

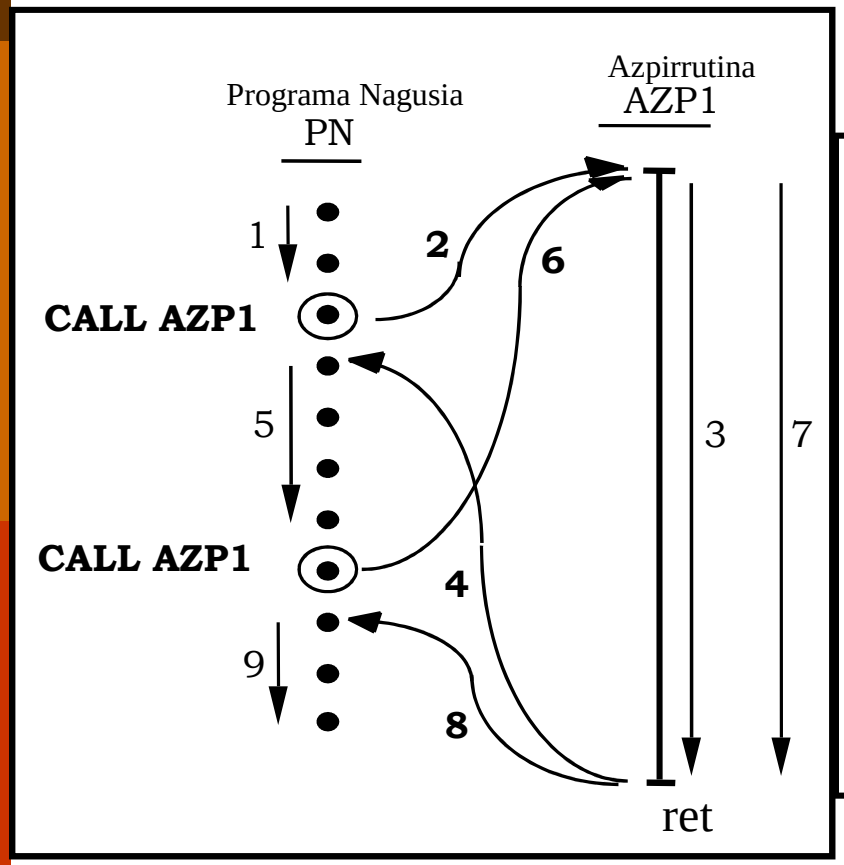
Sarrera

- Azpirrutina (funtzio edo prozedura):
  - kode-bloke batera **itzuleradun jauzia**
  - deiak/itzulerak: prozesadoreak exekutatzen dituen aginduen %3-%10 dira → hardwareak eskeini dezakeen laguntza garrantzitsua da

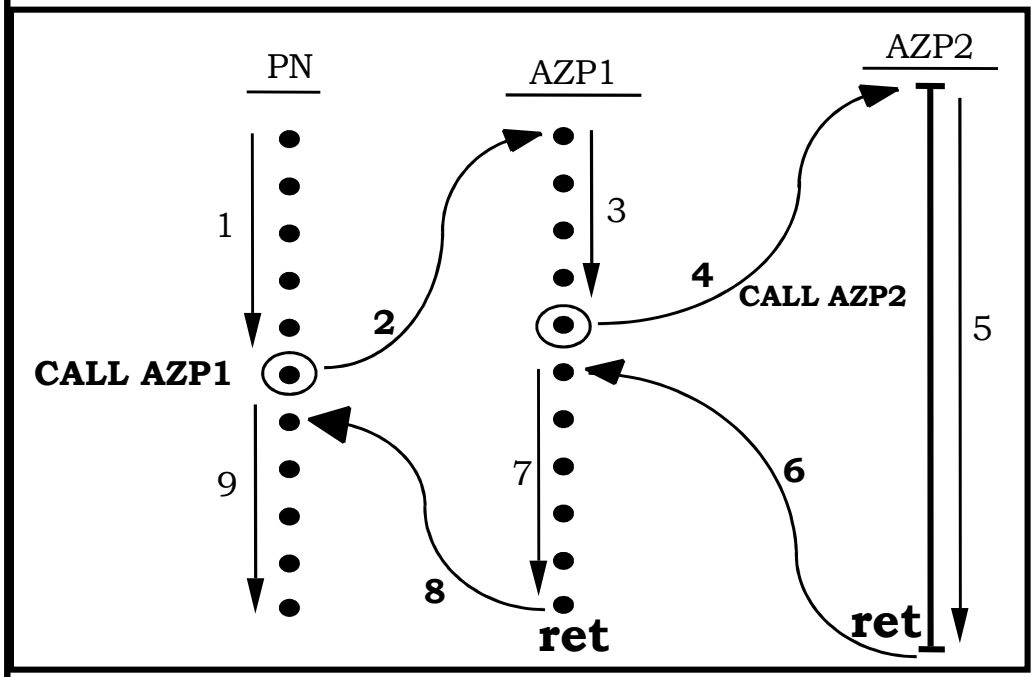


Sarrera: azpirrutinen sailkapena

- **Aktibazioaren arabera**:



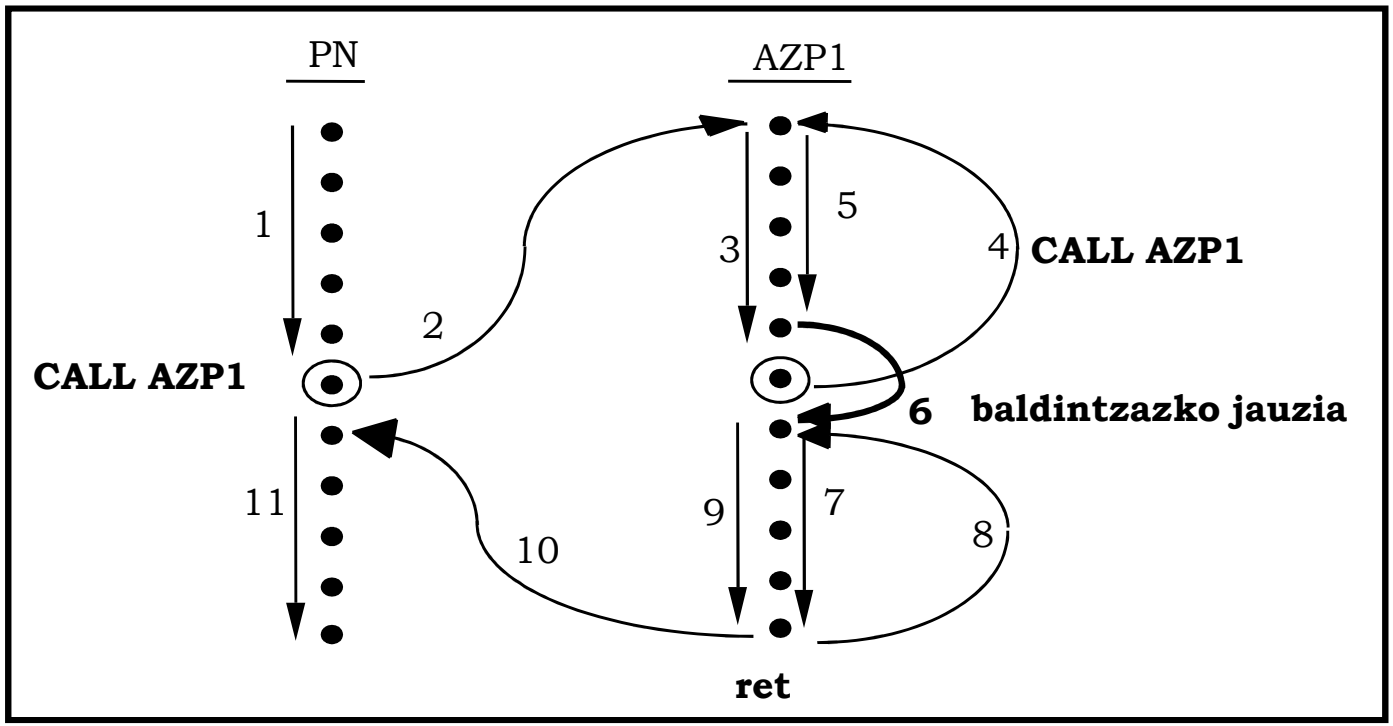
Maila bakarrekoak



Maila anitzekoak

Sarrera: azpirrutinen sailkapena

- **Aktibazioaren arabera**:



Errekurtsiboak (zuzenekoak edo zeharkakoak)

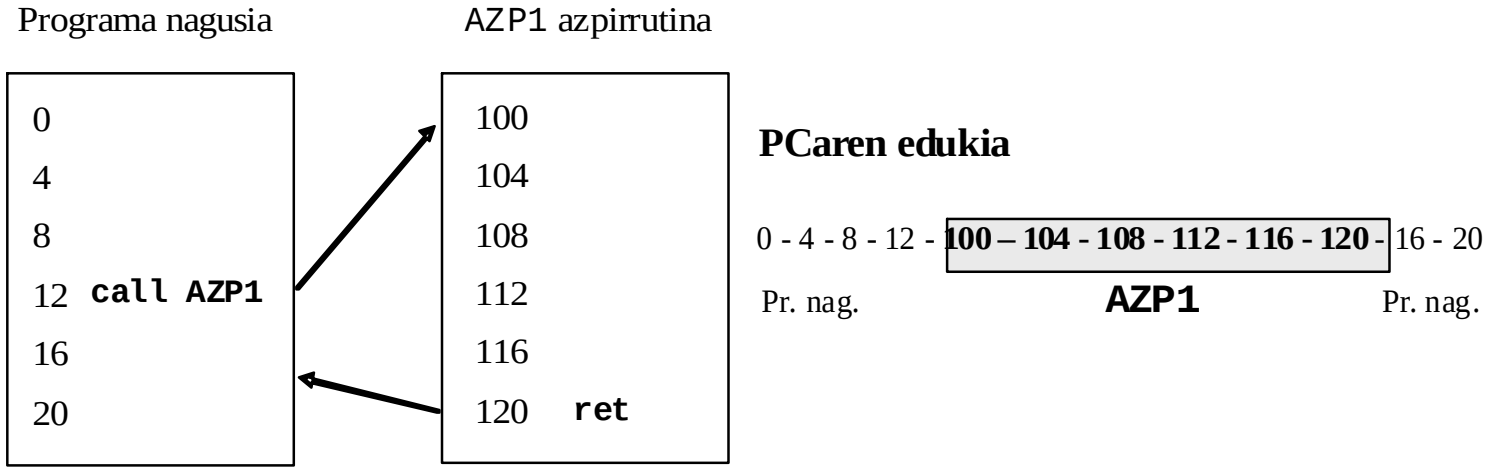
Azpirrutina baten exekuzioaren analisia

- Deia eta itzulera
- Parametro-pasatzea eta emaitzen itzulera
- Gorde eta berreskuratu programa deitzailearen egoera
- Aktibazio-blokearen kudeaketa

  - ➔ **Aktibazio-blokearen** analisia

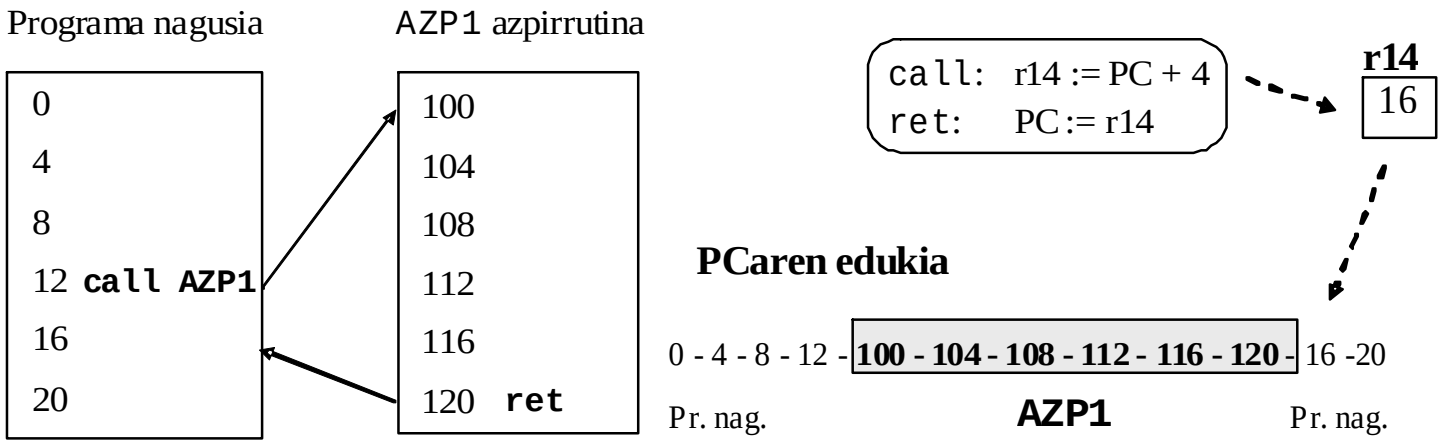
Deia eta itzulera

- Deia eta itzulera egiteko aginduak, orokorrean:
  - *CALL azpirrutina_izena:* azpirrutinari deia
    - Itzulera-helbidea gorde (@itzulera=PC_call+ 4 ARM makinan)
    - Azpirrutinaren 1. agindura jauzia (PC=@azpi=PC_call+ desp_azpi)
  - *RET:* azpirrutinatik bere programa deitzailera itzuli
    - Itzulera-helbidea berreskuratu (PC=@itzulera)



Deia eta itzulera

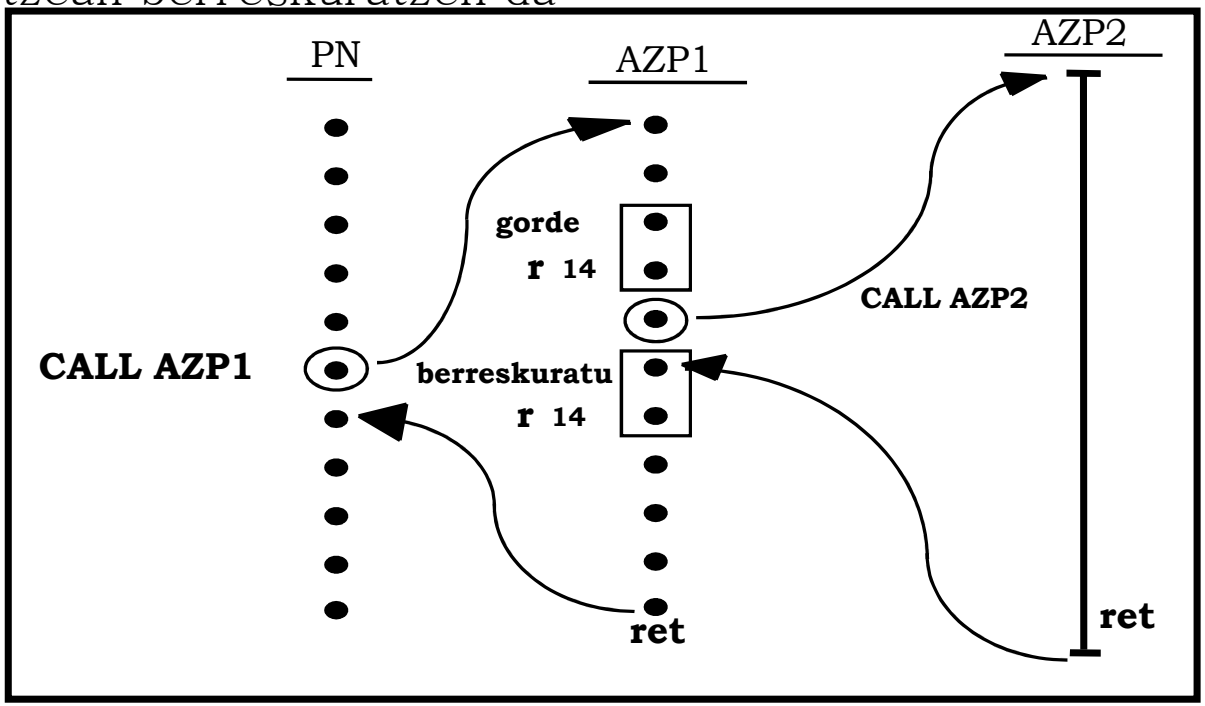
- Non gordetzen da @itzulera?
  - Memoriako toki finko batean edo azpirrutina guztientzako erregistro berean
    - Adibidez, IBM360ak R14a erabiltzen du
    - *JumpAndLink* agindua (*JAL azpirrutina*) → MIPS adibidea (R31 erregistroa)
    - *Branch and Link* agindua (BL azpirrutina) → ARM (r14 erregistroa)



ARAZOA: maila anitzeko errutinak eta errekurtsiboak

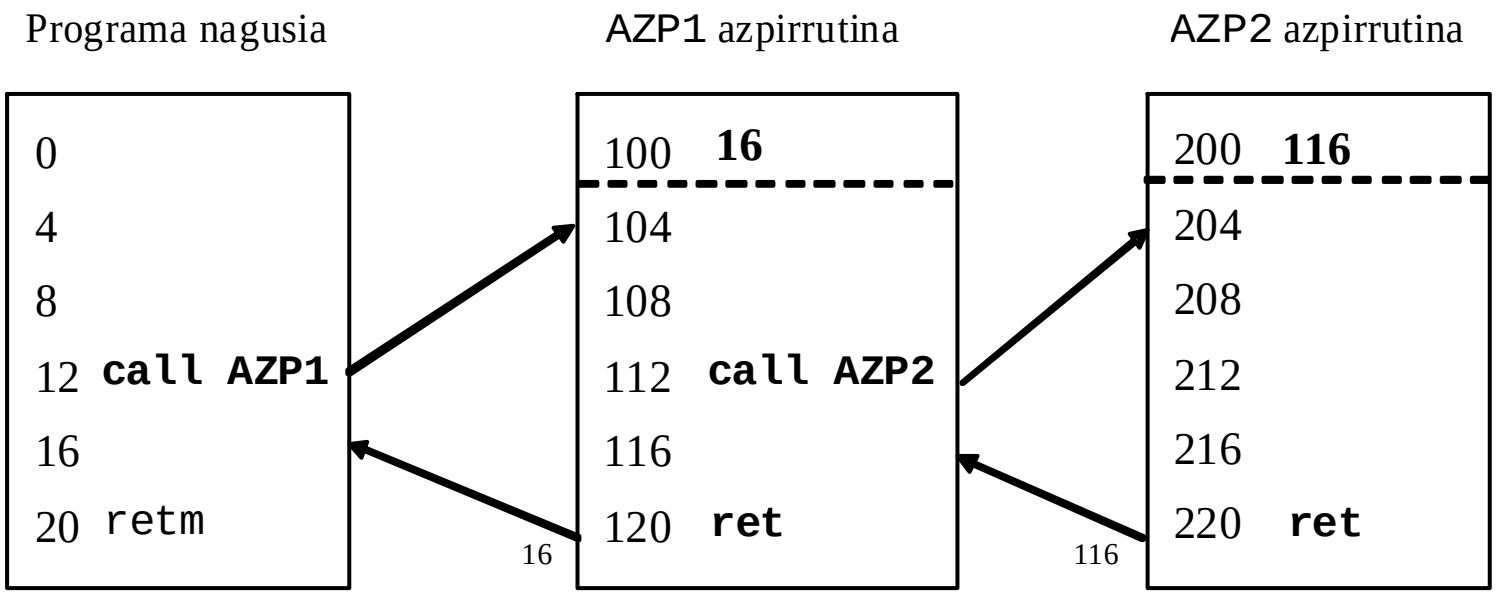
Deia eta itzulera

- Non gordetzen da @itzulera?
  - Memoriako toki finko batean edo azpirrutina guztientzako erregistro berean
    - Software ebazpena → beste azpirrutina bati deitu aurretik erregistro batean (edo memorian) gordetzen da @itzulera eta itzultzean berreskuratzen da



Deia eta itzulera

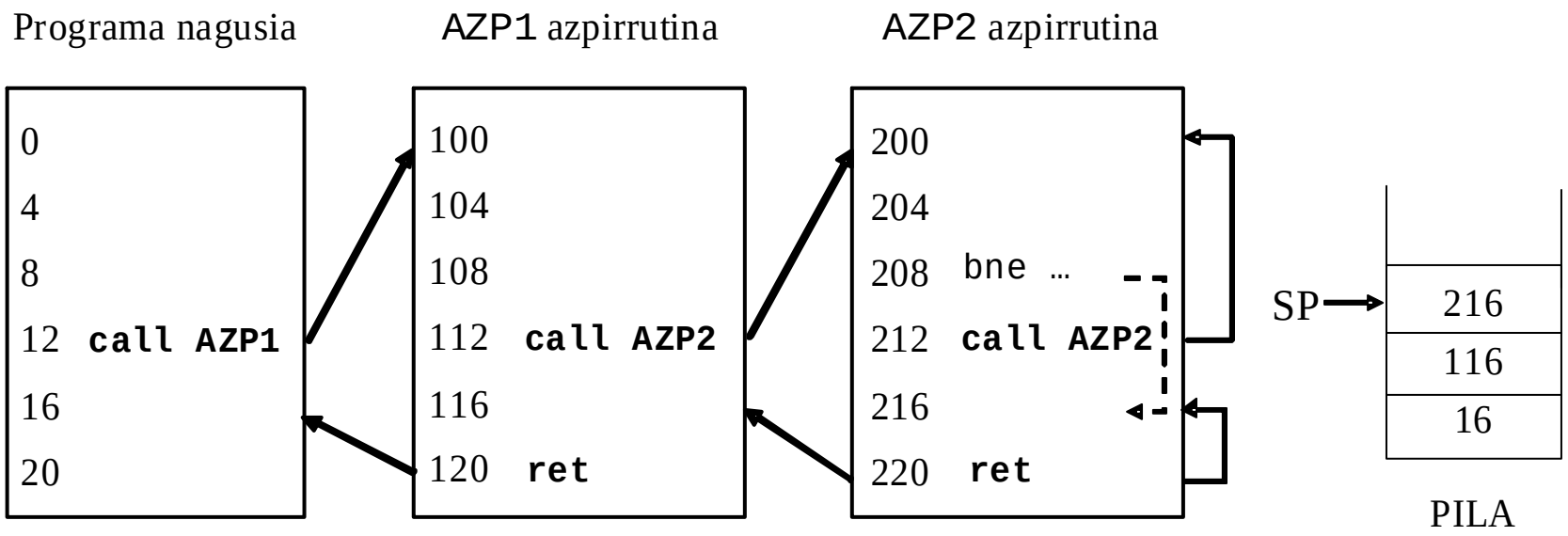
- Non gordetzen da @itzulera?
  - Azpirrutina bakoitzarentzat memoriako (edo erregistro baten) toki finko batean
    - Adibidez, HP2100 makinan @itzulera gordetzen da azpirrutinaren lehen posizioan, kodearen aurretik



ARAZOA: azpirrutina errekurtsiboak ➔ PILA baten beharra

Deia eta itzulera: PILA

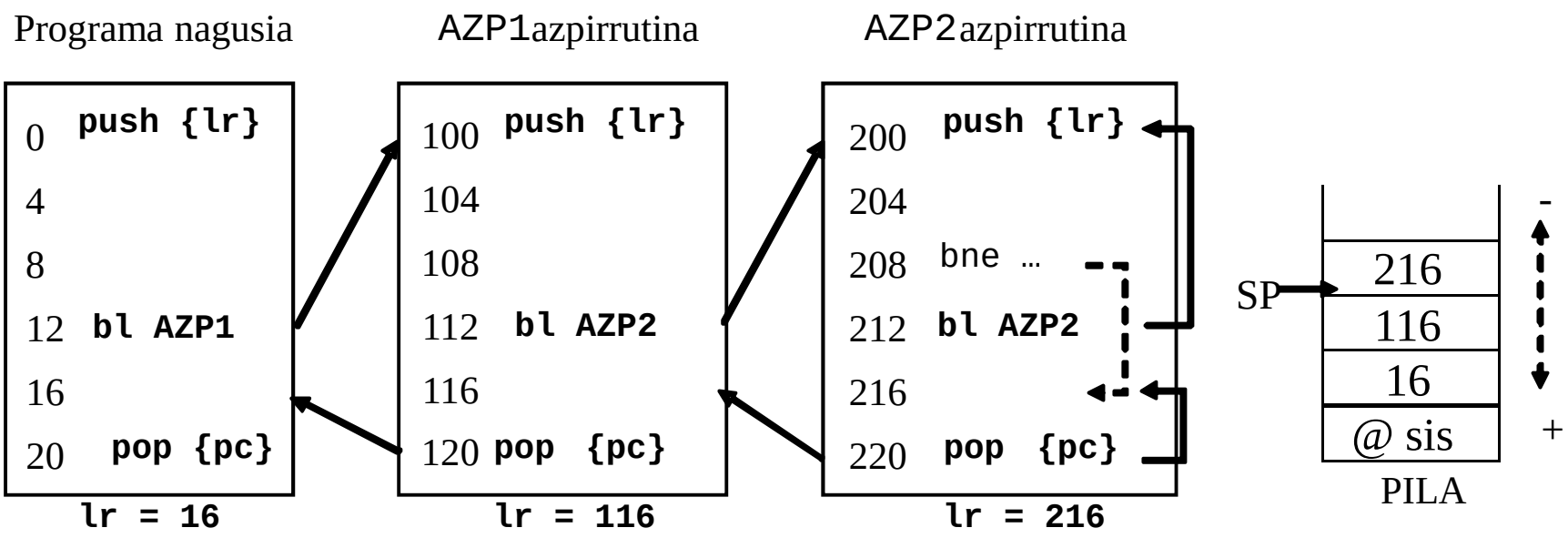
- Non gordetzen da @itzulera?
  - Pila batean
    - *CALL azpirrutina*
      - PUSH PC_Call+ 4 → pilaren tontorrean @itzulera gorde
      - PC=PC_Call+ desp_azp→ PCan kargatu azp-ren 1. aginduaren @
    - *RET*
      - POP PC → pilatik @itzulera berreskuratu eta PCan kargatu



Deia eta itzulera: ARM prozesadorea

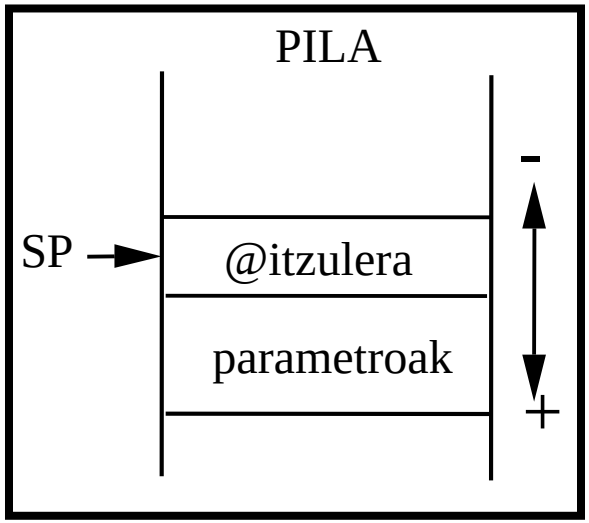
- Itzulera helbidea beti erregistro berean gordetzen da: lr edo r14 erregistroa
- Deia egiteko ondoko agindua erabiltzen da:
  - BL azpirrutina
    - *lr= PC + 4 (lr erregistroan itzulera helbidea gorde)*
    - *PC = PC + desp_azp (PCan kargatu azp-ren 1.aginduaren @)*
- Maila anitzeko azpirrutinak eta errekurtsiboak kudeatzeko PILA erabiltzen da:
  - lr erregistroaren edukia pilan gorde eta handik berreskuratu
- PILA atzitzeko:
  - push eta pop aginduak erabiltzen dira.
  - SP erakuslea r13 erregistroa da.
  - Pila helbide handietarantz hasten da.
- Ez dago RET agindurik. Itzulera helbidea pilatik berreskuratu eta PCan kargatu

Deia eta itzulera: ARM prozesadorea



Parametro-pasatzea

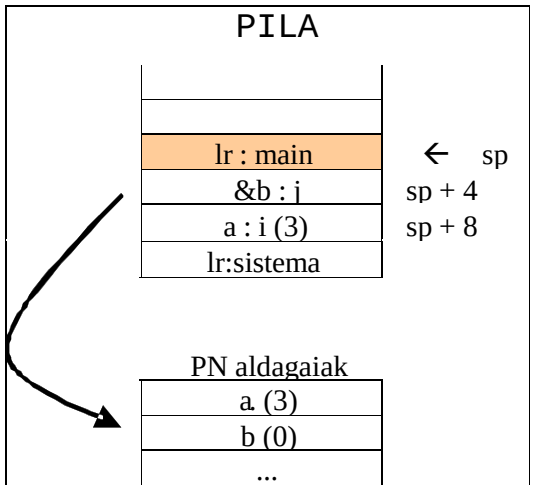
- **Parametroak**: azpirrutinak exekutatzeko behar dituen datuak
  - **Balio** bidezkoak: datuaren kopia bat pasatzen da (ezin da aldatu)
  - **Erreferentzia** bidezkoak: datuaren helbidea pasatzen da (alda daiteke)
- Programa deitzaileak parametroak pilan kokatzen ditu, azpirrutinak atzi ditzan
  - Pasatzeko: ***push {ri}*** agindua (*ri* erregistroan parametroa)
  - Atzitzeko (azpirrutinan): SParekiko (r13) helbideratze erlatiboa
- Azpirrutina bukatzean, programa deitzaileak parametroak pilatik kentzen ditu
  - Agindua: ***add sp, sp, #parametro_kopurua x 4*** (datu-tamaina)



Aktibazio-blokea

Parametro-pasatzea

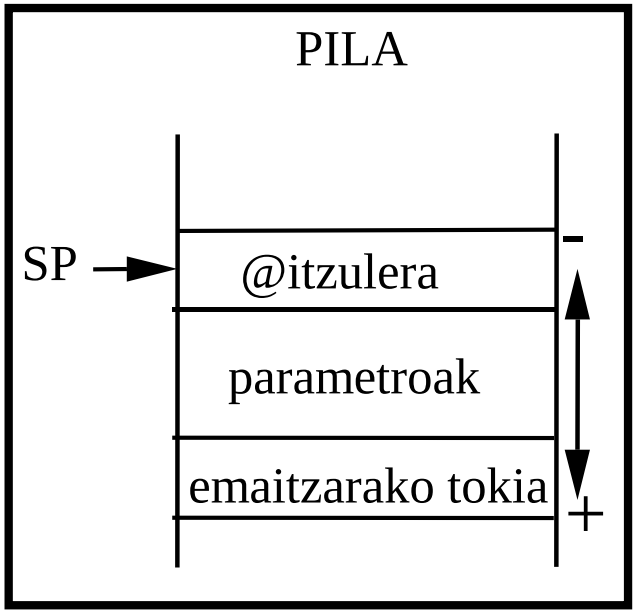
```
void AZP (int i, int *j)
{
   (*j)= i*i;
}
void main ()
{
int a=3;
int b=0;
    AZP(a, &b);
}
```




```
.code 32
.global main, __cpsr_mask

main:
  push {lr}     @itzulera helbidea pilan gorde
  ldr r1, a     @a-ren balioa r1-n gorde
  adr r0, b     @b-ren helbidea r0-n gorde
  push {r0,r1} @parametroak pilan sartu, r1,r0
  bl AZP
  add sp, sp, #8 @parametroak ezabatu pilatik
  pop {pc}       @itzulera helbide PCan kargatu
a: .word 3
b: .word 0

AZP:
  push {lr}
  ldr r3, [sp, #8]   @r3 = i (3)
  ldr r4, [sp, #8]   @r4 = i (3)
  mul r5,r3,r4
  ldr r6, [sp, #4]   @r6 = @b = j
  str r5, [r6]
  pop {pc}
```

Emaitzen itzulera

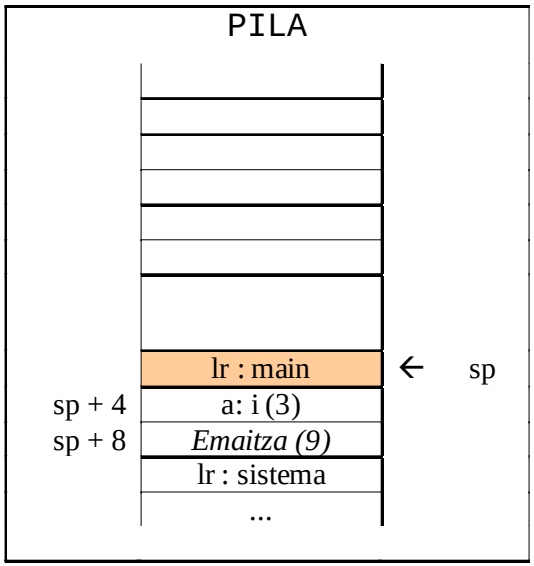
- Azpirrutina funtzioa bada, programa deitzaileak pilan tokia gorde behar du azpirrutinak emaitza itzuli ahal izateko
- Tokia erreserbatzen da azpirrutinari parametroak pasatu aurretik
  - Agindua: ***sub sp,sp,#4 (datu-tamaina)***
- Azpirrutina bukatzean, programa deitzaileak pilan dagoen emaitza jasotzen du parametroak ezabatu ondoren
  - Agindua: ***pop {rh}*** (emaitza *rh* erregistroan)



Aktibazio-blokea

Parametro-pasatzea / emaitzen itzulera

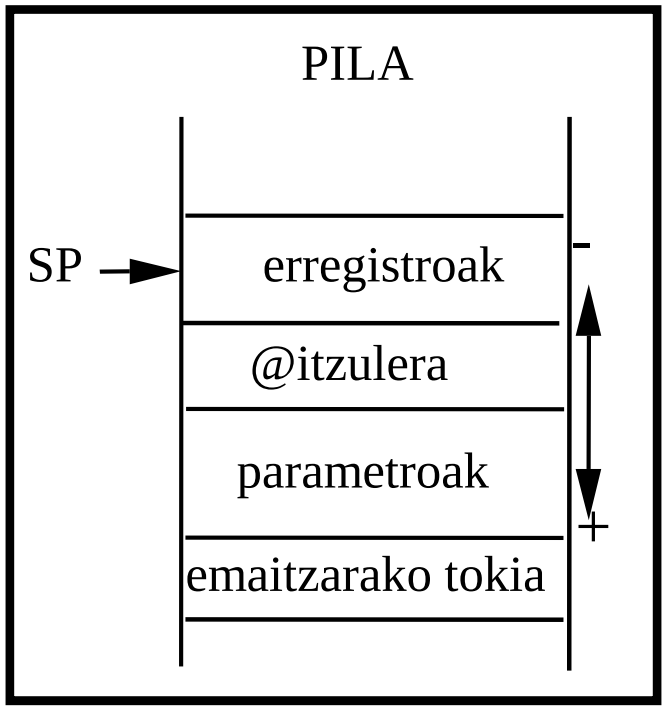
```
int AZP (int i)
{
    return (i*i);
}
void main ()
{
int a=3;
int ema;
    ema=AZP(a);
}
```




```
.code 32
.global main, __cpsr_mask

main:
  push {lr}
  ldr r0, a
  sub sp, sp, #4    @emaitzarako tokia gorde pilan
  push {r0}         @parametroa pasa
  bl AZP
  add sp, sp, #4    @parametroa kendu pilatik
  pop {r0}          @emaitza jaso pilatik
  str r0, ema
  pop {pc}
a: .word 3
ema: .word 0

AZP:
  push {lr}
  ldr r3, [sp, #4]
  ldr r4, [sp, #4]
  mul r5,r3,r4
  str r5, [sp, #8] @emaitza utzi pilan
  pop {pc}
```

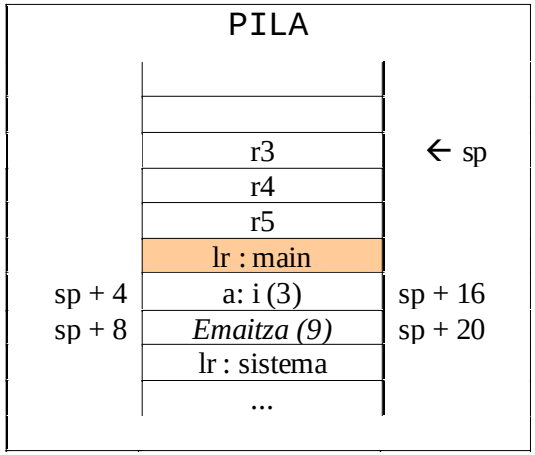
Programa deitzailearen egoera gorde eta berreskuratu

- Aurreko adibidean, azpirrutinak eta programa deitzaileak erregistro desberdinak erabiltzen dituzte
  - Ezagutu behar zeintzuk diren azpirrutinak erabili behar ez dituenak
  - Erregistroen kopurua mugatuta
- Konponbidea: **azpirrutinaren exekuzioa gardena** egin bere programa deitzailearekiko
- **Aukerak**:
  - Azpirrutinak, exekuzioaren hasieran, pilan gordetzen ditu aldatu behar dituen erregistroen balioak eta exekuzioaren bukaeran berreskuratzen ditu →HAU
  - Azpirrutinari deia egin aurretik, programa deitzaileak berak erabiltzen dituen erregistroen balioak gordetzen ditu eta azpirrutina exekutatzen bukatzean berreskuratzen ditu



Aktibazio-blokea

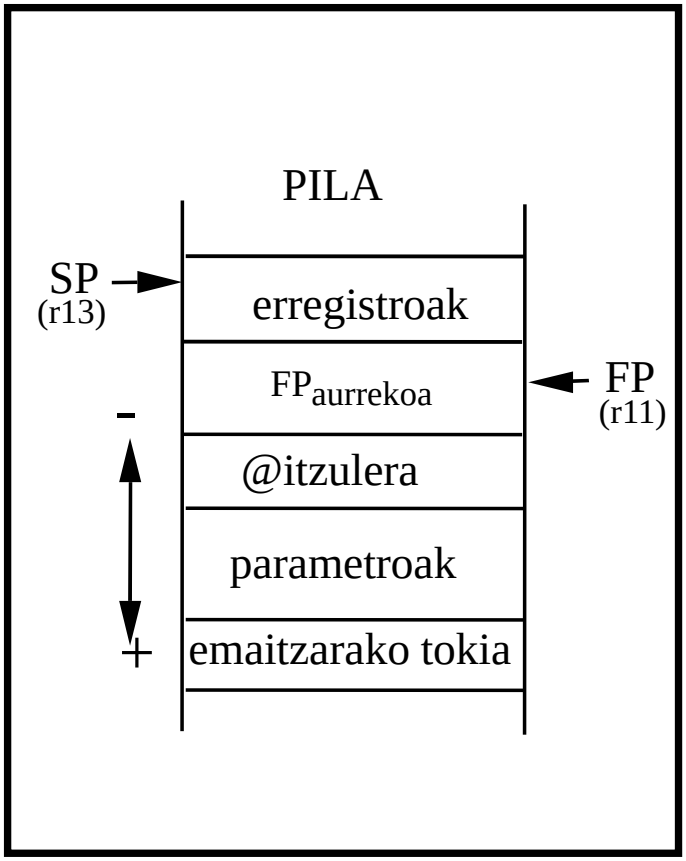
Programa deitzailearen egoera gorde eta berreskuratu



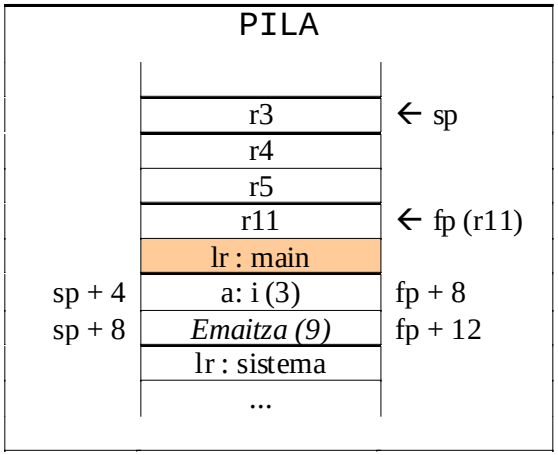

```
.code 32
.global main, __cpsr_mask

main:
   push {lr}
   ldr r0, a
   sub sp, sp, #4
   push {r0}
   bl AZP
   add sp, sp, #4
   pop {r0}
   str r0, ema
   pop {pc}
a: .word 0x00000003
ema: .word 0x00000000

AZP:
   push {r3,r4,r5,lr}
   ldr r3, [sp, #16]
   ldr r4, [sp, #16]
   mul r5,r3,r4
   str r5, [sp, #20]
   pop {r3,r4,r5,pc}
```

Aktibazio-blokearen kudeaketa

- SParen balioa exekuzio-denboran aldatzen doa: gorde behar diren erregistroen menpekoa
  - → **SPa erabiliz AB atzitzea zaila**
- Konponbidea: AB atzitu SParen baliotik at, azpirrutinaren exekuzioan zehar ABaren helbide finko baten erakuslearekin
  - → **FP (Frame Pointer) erakuslea: r11 erregistroa**
- FP erakuslea azpirrutinaren exekuzioaren hasieran finkatzen da
- ABaren atzipen guztiak FP erabiliz gauzatzen dira
  - → **FParekiko heldiberatze erlatiboa**
- FP erregistro bat da eta, beraz, azpirrutinaren hasieran gorde behar da



Programa deitzailearen egoera gorde eta berreskuratu




```
.code 32
.global main, __cpsr_mask

main:
    push {lr}
    ldr r0, a
    sub sp, sp, #4
    push {r0}
    bl AZP
    add sp, sp, #4
    pop {r0}
    str r0, ema
    pop {pc}
a: .word 3
ema: .word 0
```

```
AZP:
    push {r11,lr}
    mov r11, sp
    push {r3,r4,r5}
    ldr r3, [r11, #8]
    ldr r4, [r11, #8]
    mul r5,r3,r4
    str r5, [r11, #12]
    pop {r3,r4,r5,r11,pc}
```

2.gaia.- Agindu multzoa

1. Makina lengoaia eta mihiztadura lengoaia
2. Azpirrutinak
3. Aginduen formatua
   - Helbideratze moduak
4. Makinen sailkapena aginduen formatuaren arabera

Aginduen formatua. Helbideratze moduak

Makina lengoaiaren ezaugarriak:

- Agindu multzoa
- Aginduen formatua
- Eragigai kopurua
- Helbideratze-moduak
- Datu-motak eta beraien formatua

Aginduen formatua

Eragikea-kodea	Helburu-eragigaia	….	Iturburu-eragigaia

Formatua: aginduak izan behar duen informazioa bit segida batean kodetzen den modua

- *eragiketa-kodea: a*ginduak burutu behar duen eragiketa.
- Iturburu eragigaiak: beraien balio edo kokapena
- Helburu eragigaia: emaitza non utzi behar den

Aginduen luzera

- **Luzera finkoko** formatua: agindu guztiek tamaina bera dute, hau da, bit-kopuru bera erabiltzen da agindu guztiak kodetzeko
- **Luzera aldakorreko** formatua: agindu formatu desberdinak daude tamaina desberdinekoak direnak. Aginduak jarraitzen duen formatua zein den jakiteko informazioa behar da.

Eragiketa-kodearen formatua

- **Luzera finkoko eragiketa-kodea:**
  - Eragiketa-kodearen tamaina edo bit-kopura berdina da agindu guztientzat. Bere luzera, l, kodetu behar den agindu-kopuruaren araberakoa da:

$$l = \lceil \log_2 (agindu \ - kopurua \ ) \rceil$$

- **Luzera aldakorreko eragiketa-kodea:**
  - Agindu guztien eragiketa-kodeak ez du luzera berdina. Helburua eragigaien luzera eragiketa-kodearen luzerarekin orekatzea da, bataz-beste agindu tamaina txikiagoa lortzeko edo eragigai gehiago adierazi ahal izateko.
    - Kode-zabaldua (PDP-11)
    - Huffman kodeketa(MC68000)

Eragigaien formatua: helbideratze moduak

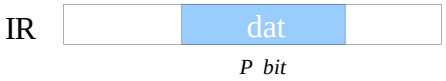
Helbideratze-moduek eragigaiak nola lortu eta emaitza non gorde adierazten digute. Eragigaien formatuan datu horiek atzitzeko modua adierazi behar da.

Helbideratze moduak: atzipen-denbora desberdinak eta bit-kopuru desberdina

- Berehalakoa
- Erregistro bidezko zuzenekoa
- Absolutua
- Erregistro bidezko zeharkakoa
- Erlatiboa: oinarri erregistroa + desplazamendua
- Memoria bidezko zeharkakoa
- Indexatua
- Oinarri-erregistroa + indize-erregistroa

Berehalakoa

Eragigaia aginduan bertan dagoen konstante bat da



ARM: mov r0, **#512** (hamartarra)
mov r0, **#0x8A** (hamaseitarra)

12 bitekin kodetzen dira ARM-an:

- IR (4 bit): IR*2 aldiz errotatuko da eskuinera IN eremua
- IN (8 bit): 8 biteko balio bat da

$$[0...255]*2^{2n}$$

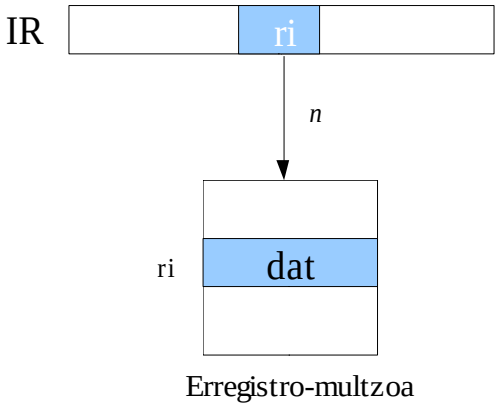
Baldintza	00	I	Eragiketa	S	Ri1	Rh	**2º Eragigaia**
1110	00	1	0100	0	0001	0010	0000 | 00001100
Beti			ADD		R1	R2	**#12**

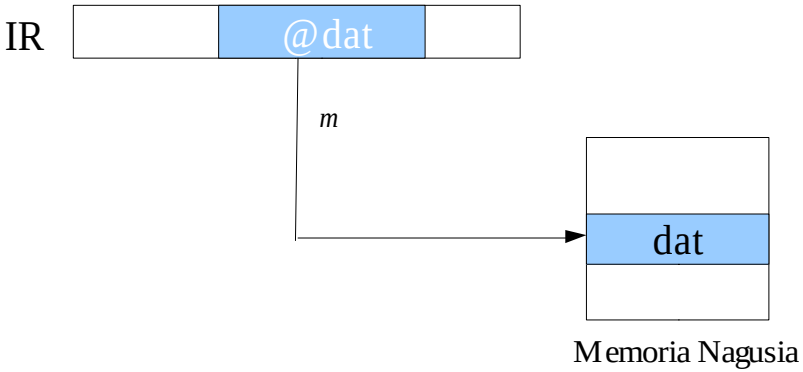
add r2,r1,***#12***

Erregistro bidezko zuzenekoa, absolutua

Erregistro bidezko zuzenekoa

datua = EM [erregistro zenbakia]





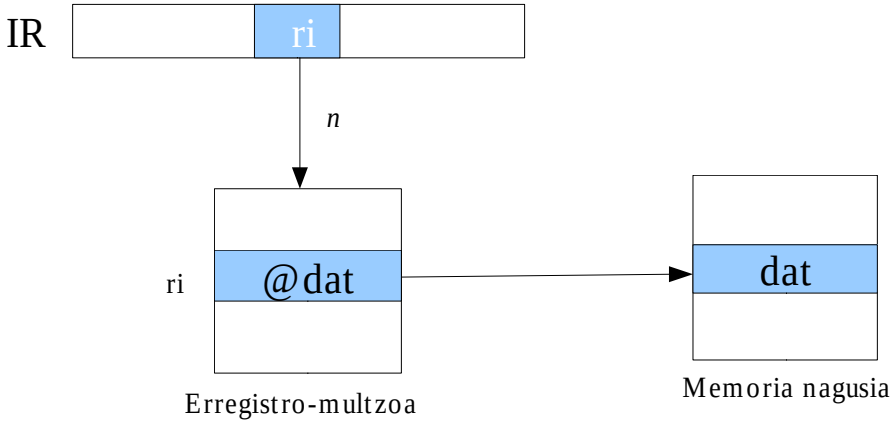
Absolutua

datua = M[helbidea]

Erregistro bidezko zeharkakoa, erlatiboa

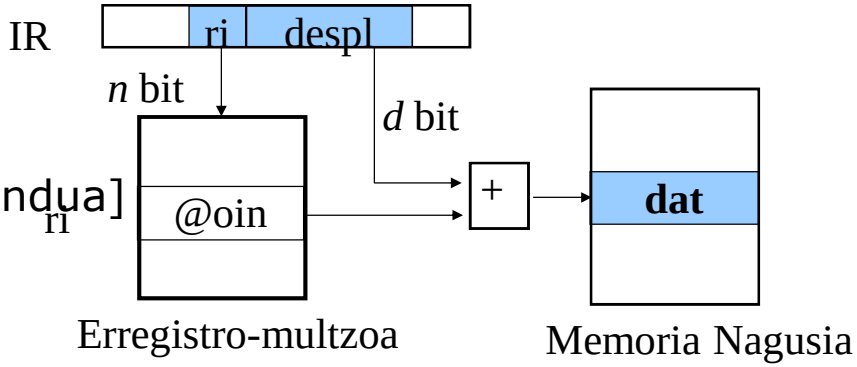
datua = M[EM[erregistro zenbakia]]

Erregistro bidezko zeharkakoa



Erlatiboa

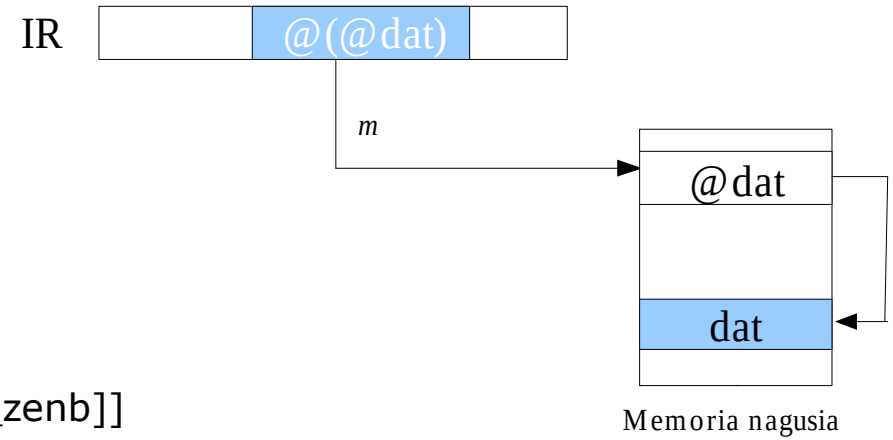
datua = M[EM[erregistro zenb.] + desplazamendua]



Memoria bidezko zeharkakoa, indexatua

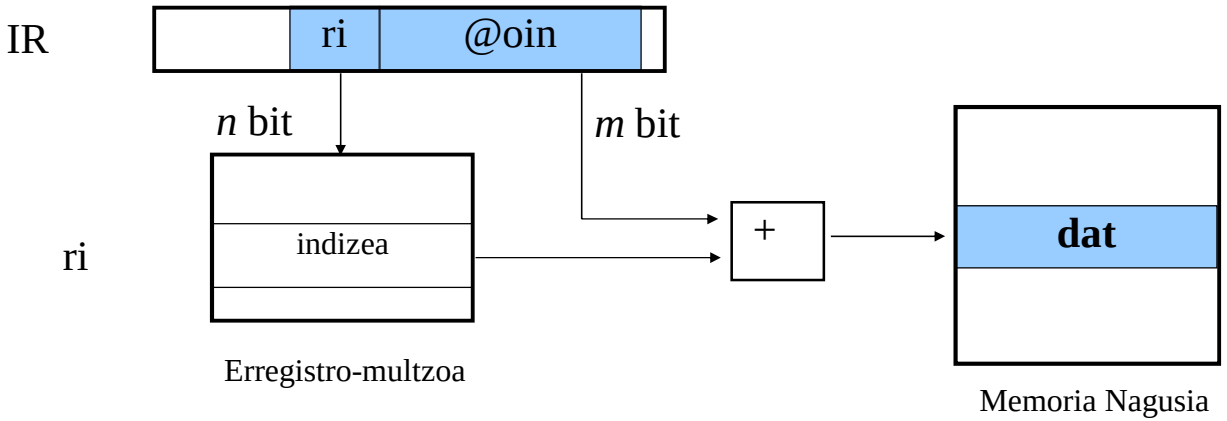
Memoria bidezko zeharkakoa

datua = M[M[helbidea]]



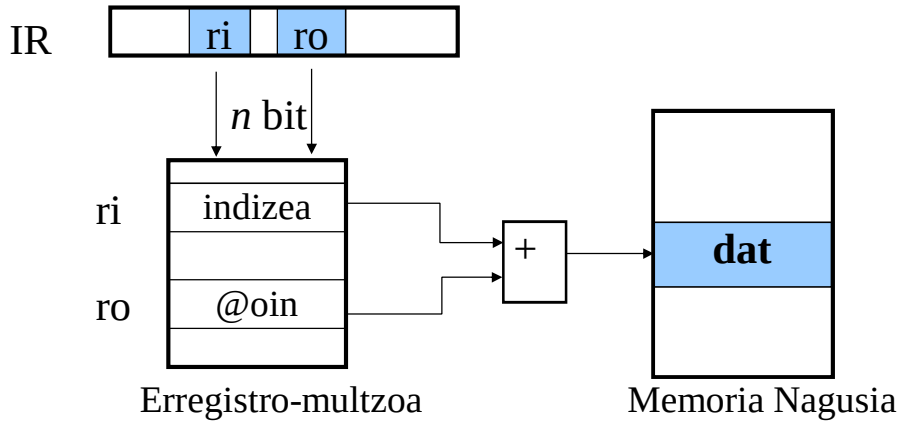
Indexatua

dato = M[helbidea + EM[erregistro_zenb]]



Oinarri erregistroa + indize erregistroa

datua = M[EM[o.erreg.] + EM[i.erreg]]



Pre/post indexazioa eta berridazketa

Memoria atzitzeko helbidea kalkulatu behar denean batuketa bat eginez (h.m erlatiboa eta o.e + i.e) ondoko aukerak dauzkagu:

- ***Pre-indexazioa*** erabiltzen denean bi datuak kortxete artean idazten dira eta emaitza berridaztea nahi izanez gero, bukaeran harridura ikurra jarri behar da.

ldr r2, ***[R3, R4]***	*@ r2=Mem(r3+r4)*
ldr r2, ***[R3, R4]!***	*@ r2=Mem(r3+r4) eta r3=r3+r4*
ldr r2, ***[R3, #4]***	*@ r2=Mem(r3+#4)*
ldr r2, ***[R3, #4]!***	*@ r2=Mem(r3+#4) eta r3=r3+#4*

- ***Post-indexazioa*** erabiltzen denean beti egiten da berridazketa, baina kalkulua memoria atzitu ondoren egiten da.

ldr r2, ***[R3], R4***	*@r2=Mem(r3) eta r3=r3+r4*
ldr r2, ***[R3], #4***	*@r2=Mem(r3) eta r3=r3+#4*

Pre/post indexazioa eta berridazketa

Adibidea

```
for (i=0; i< 9; ++)
          B[i] = A[i] + A[i+1];

          .code 32
          .global main, __cpsr_mask
          B: .word 0,0,0,0,0,0,0,0,0
          A: .word 5, 4, 3,7,8,6,7,5,1,2
          main:
          mov r0, #0
          adr r1, B
          adr r2, A
     FOR: cmp r0, #9
          beq buk
          ldr r3,[r2]
          ldr r4, [r2, #4]!
          add r5, r3, r4
          str r5, [r1], #4
          add r0, r0,#1
          b FOR
     buk: mov pc, lr
```