SISTEM MIKROPROSESOR

PERTEMUAN 4

Mochamad Fajar Wicaksono, S.Kom., M.Kom.

PERCABANGAN, STACK, SUBRUTIN DAN DELAY

PERCABANGAN

Dua macam percabangan:

- Percabangan tak bersyarat (unconditional branch)
- Percabangan bersyarat (conditional branch)

PERCABANGAN TAK BERSYARAT

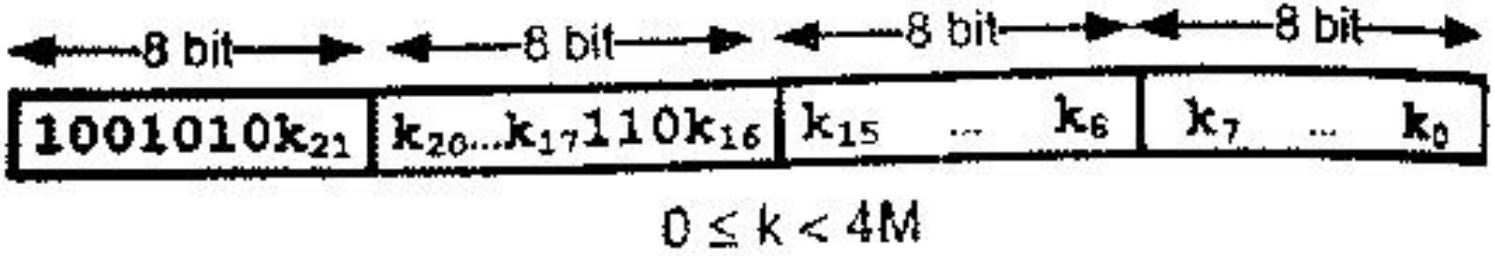
- JMP
- RJMP
- IJMP

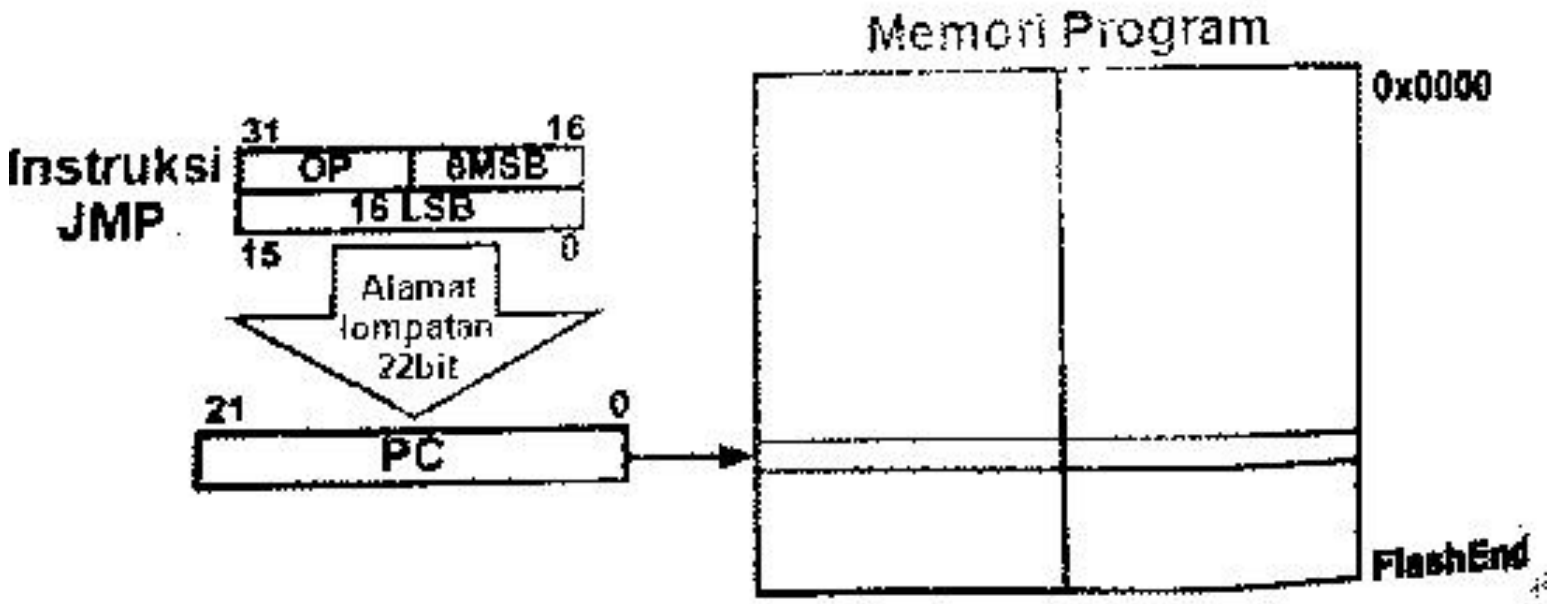
PERCABANGAN TAK BERSYARAT

JMP (JUMP)

- Instruksi ini panjangnya 32 bit : 10 bit opcode dan 22 bit sebagai alamat.
- Alamat operand JMP sebanyak 22bit menghasilkan lompatan 4M word (alamat $000000 s/d $3FFFFF).
- Kekurangan: ruang kode butuh 2 word
- Contoh: **JMP lompat**

PERCABANGAN TAK BERSYARAT

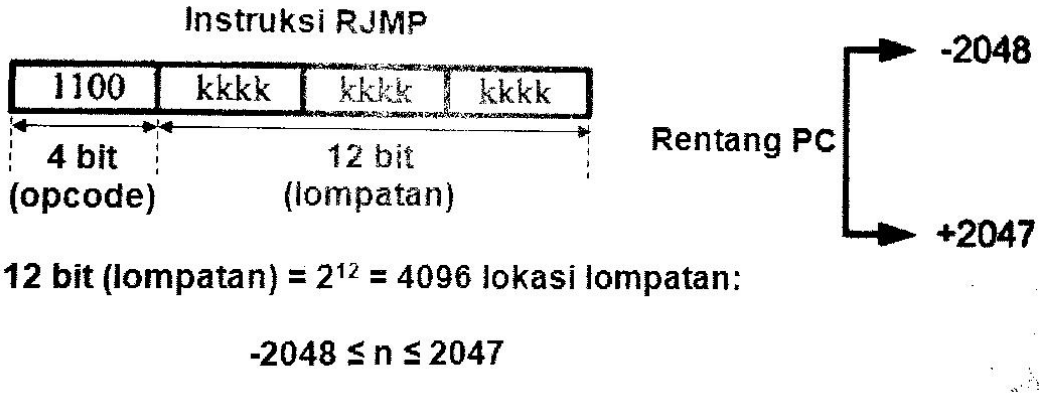




PERCABANGAN TAK BERSYARAT

RJMP (RELATIVE JUMP)

- Panjang 16bit 4 bit opcode dan 12 bit alamat.
- Lompatan sebesar 4K word (alamat relative $000 s/d $FFF).
- Contoh: **rjmp ulang**

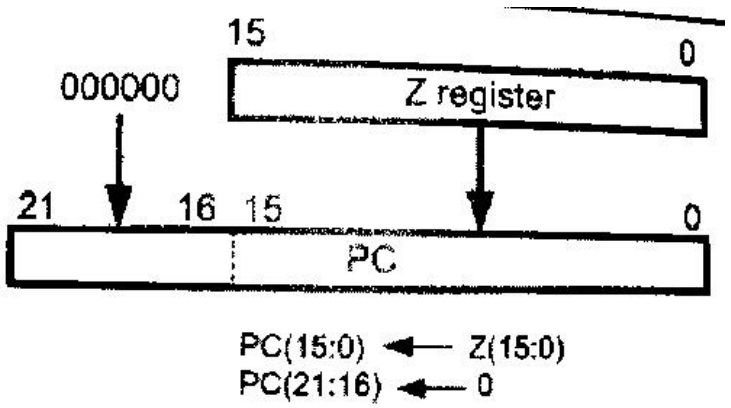


12 bit (lompatan) = 2^{12} = 4096 lokasi lompatan:

$-2048 \leq n \leq 2047$

PERCABANGAN TAK BERSYARAT

- IJMP (INDIRECT JUMP ke Z)
- Melakukan lompatan ke alamat yang ditunjuk oleh pasangan register indeks Z.
- Dengan lebar Z 16 bit mengizinkan lompatan 64K word.
- Instruksi ini baik untuk lompatan dalam perhitungan alamat atau alamat dari suatu lookup table.



CONTOH:

LDI ZL(ULANG)

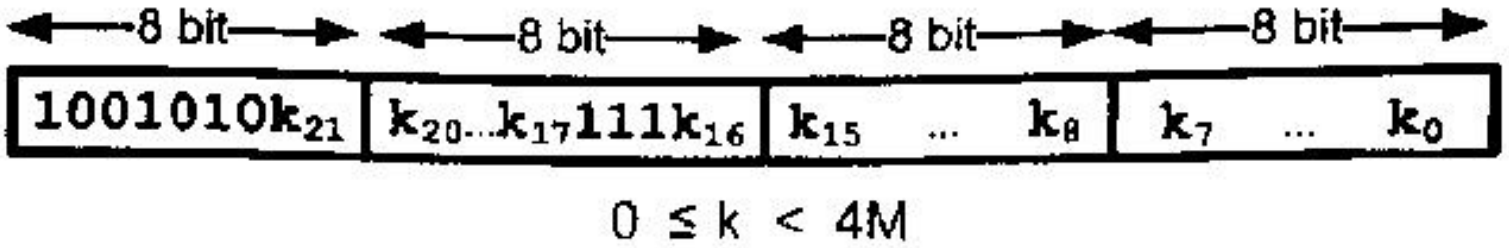
LDI ZH(ULANG)

IJMP

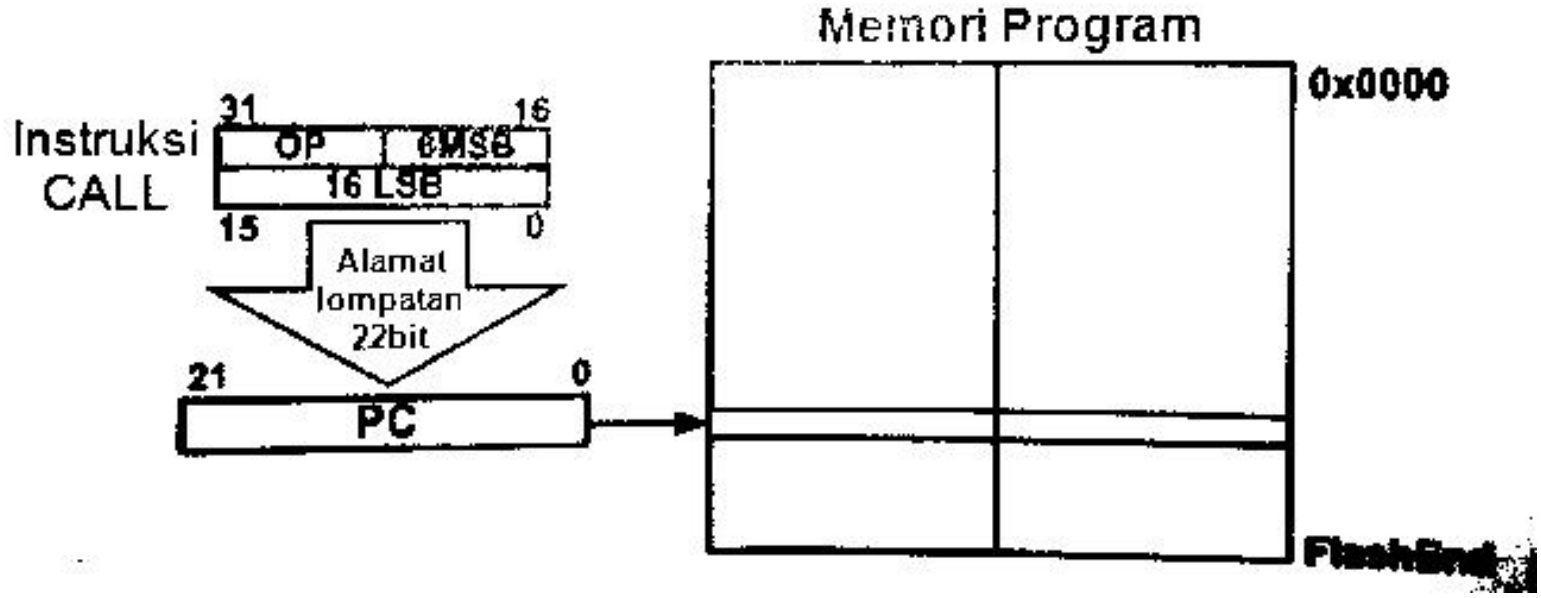
INSTRUKSI PEMANGGILAN SUBRUTIN (CALL)

- **CALL (long call)**
- **RCALL (relative call)**
- **ICALL (indirect call to Z)**

INSTRUKSI PEMANGGILAN SUBRUTIN (CALL)



$0 \leq k < 4M$

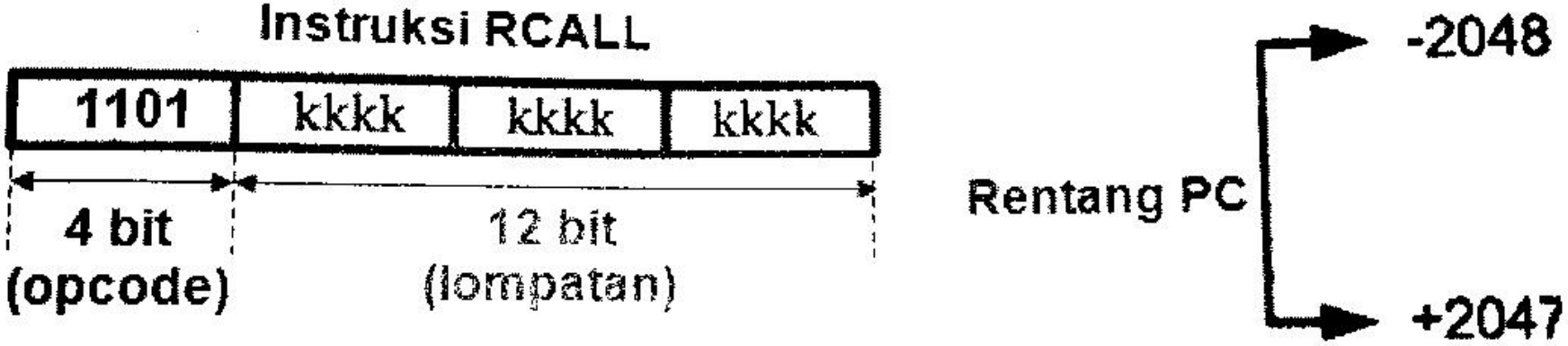


INSTRUKSI PEMANGGILAN

CALL (LONG CALL)

- Termasuk dalam lompatan tak bersyarat.
- Instruksi ini akan beralih ke sebuah subrutin yang sudah ditetapkan sebelumnya dan diakhiri dengan instruksi kembali (return).
- Panjang instruksi 32 bit opcode 10 bit dan alamat lompatan ke sub rutin 22bit.
- Jangkauan alamat $000000 - $3FFFFF
- Contoh: **CALL Subrutin**

INSTRUKSI PEMANGGILAN



12 bit (lompatan) = 2^{12} = 4096 lokasi lompatan:

$$-2048 \leq n \leq 2047$$

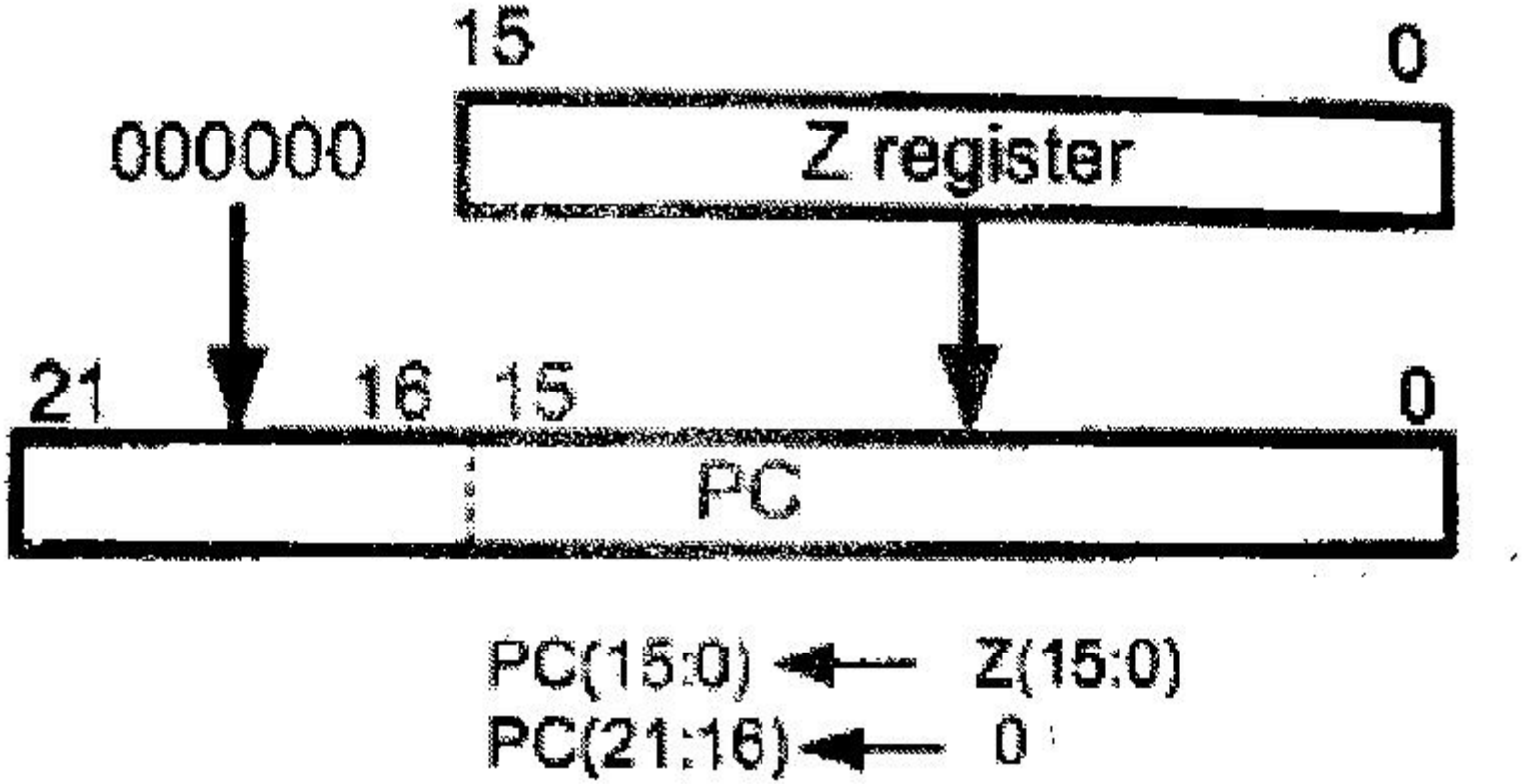
INSTRUKSI PEMANGGILAN

RCALL (RELATIVE CALL TO SUBROUTINE)

- Lompatan tidak melebihi dari 4Kword.
- Panjang instruksi 16bit 4bit opcode dan 12 bit alamat lompatan.
- Rentang lompatan -2048 s/d +2047
- Contoh: **RCALL subprogram**

INSTRUKSI PEMANGGILAN

ICALL



INSTRUKSI PEMANGGILAN

ICALL (INDIRECT CALL)

- Instruksi ini dapat menjangkau alamat sampai 64K.
- **Register Z** digunakan untuk menetapkan alamat target lompatan.
- Ketika diekseskusi maka alamat instruksi setelah kembali (setelah call) didorong kedalam Stack dan isi register Z dimasukkan ke PC.

- Contoh: ldi ZL,low(ulang)
  ldi ZH,high(ulang)
  ICALL

INSTRUKSI PEMANGGILAN

RET (RETURN)

- Keluar dari subrutin atau kembali instruksi tepat dibawah baris instruksi pemanggil subrutin.
- Ketika instruksi RET dieksekusi maka lokasi teratas stack disalin kembali ke dalam Program Counter dan stack pointer di incrementkan.
- Ketika instruksi CALL dieksekusi maka alamat instruksi yang berada dibawah instruksi CALL di push kedalam stack.

LOMPATAN BERSYARAT

- Percabangan ini berdasarkan pada register status mikrokontroler (SREG).
- Semua instruksi lompatan bersyarat mempunyai lebar 2byte.

LOMPATAN BERSYARAT

SBIC (skip if bit in I/O register is cleared).

- Intruksi ini akan memeriksa apakah sebuah bit pada I/O register dalam keadaan clear. Jika ya maka lompati (skip) satu instruksi dibawahnya.
- Contoh: **sbic PORTA,2**

LOMPATAN BERSYARAT

SBIS (skip if bit in I/O register is set)

- Instruksi ini memeriksa apakah sebuah bit pada I/O register dalam keadaan set. Jika ya maka lompati satu instruksi dibawahnya.
- Contoh: **sbis PORTA,3**

LOMPATAN BERSYARAT

SBRC (skip if bit in register is cleared)

- Instruksi ini memeriksa apakah sebuah bit pada suatu register dalam keadaan clear. Jika ya maka lompati satu instruksi dibawahnya.

Contoh:

Sbrc r0,7 ; lompati satu perintah jika bit 7 r0 clear

LOMPATAN BERSYARAT

SBRS (skip if bit in register is set)

- Instruksi ini memeriksa apakah sebuah bit pada suatu register dalam keadaan set. Jika ya maka lompati satu instruksi dibawahnya.

Contoh:

Sbrs r2,4 ; lompati satu perintah jika bit 4 r2 set

LOMPATAN BERSYARAT

BREQ (BRANCH IF EQUAL)

- Instruksi ini merupakan instruksi conditional relative branch.
- Instruksi ini akan lompat ke alamat yang ditunjuk bila antara dua register atau antara register dengan konstanta yang dibandingkan **sama**.
- Dengan kata lain akan lompat ketika flag Z=1
- Instruksi ini umumnya dipasangkan dengan instruksi CP, CPI, SUB atau SUBI.

- **Contoh: cp r0,r1**
  breq sama

LOMPATAN BERSYARAT

BRNE (BRANCH IF NOT EQUAL)

- Instruksi ini akan lompat ke alamat yang ditunjuk bila antara dua register atau antara register dengan konstanta yang dibandingkan **tidak sama**.
- Dengan kata lain akan lompat ketika flag Z=0
- Instruksi ini umumnya dipasangkan dengan instruksi CP, CPI, SUB atau SUBI.

- **Contoh:** **cpi r16,5**
  brne tidaksama

LOMPATAN BERSYARAT

BRLO (BRANCH IF LOWER (UNSIGNED))

- Instruksi ini akan lompat ke alamat yang ditunjuk bila antara dua register atau antara register dengan konstanta yang dibandingkan **lebih kecil**.
- Instruksi ini umumnya dipasangkan dengan instruksi CP, CPI, SUB atau SUBI.

- **Contoh: cpi r19,$10**

  brlo lebihkecil

LOMPATAN BERSYARAT

BRSH (BRANCH IF SAME OR HIGHER (UNSIGNED))

- Instruksi ini akan lompat ke alamat yang ditunjuk bila antara dua register atau antara register dengan konstanta yang dibandingkan **sama atau lebih besar**.
- Instruksi ini umumnya dipasangkan dengan instruksi CP, CPI, SUB atau SUBI.

- **Contoh: cpi r19,4 ;bandingkan r19 dengan 4**
  brsh highsm

SELEKSI KONDISI

IF-THEN-ELSE

IF kondisi benar
THEN
    Eksekusi branch statements
ELSE
    Eksekusi false-branch statement
END_IF

CONTOH

Tuliskan kode assembly struktur keputusan berikut:

IF R17<0 THEN

Kirim byte 0xaa ke portA

ELSE

Kirim byte 0x55 ke portA

END_IF

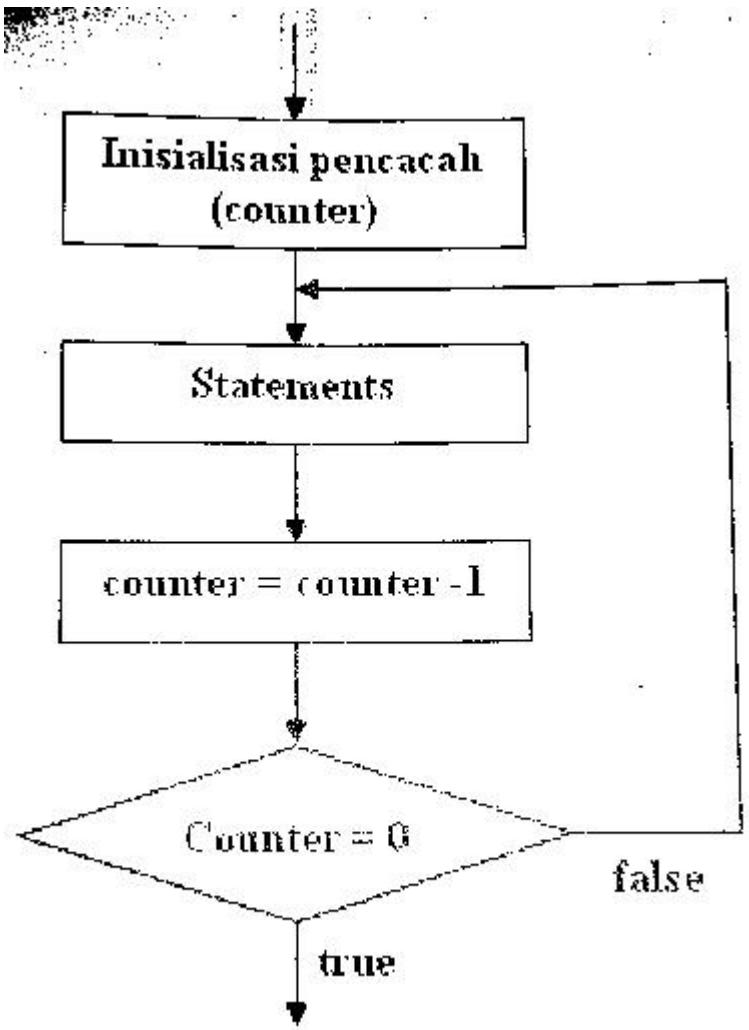
```
.Include “m32def.inc”
.org 0x0000
Rjmp utama
Utama:
Ldi r16,low(ramend)
Mov spl,r16
Ldi r16,high(ramend)
Mov sph,r16

Ldi r16,0xff ;port a output
Out DDRA,r16

;----IF R17<0--------
    Cpi r17,0
    Breq ELSE_IF
;----THEN-----------
    Ldi r16,0xaa
    Out porta,r16
    Rjmp END_IF
:ELSE_IF:
    Ldi r16,0x55
    Out porta,r16
END_IF:
    Stop: rjmp Stop
```

PENGULANGAN FOR

- Banyaknya pengulangan tergantung pada kondisi.

```
FOR jumlah  loop_count DO
        Statements
END_FOR
```

SKEMA


Inisialisasi pencacah
(counter)
Statements
counter = counter -1
Counter = 0
false
true

CONTOH

```
Ldi r16,0
Loop:
        out PORTB,r16
        Inc r16
        cpi r16,10
        Brne loop
```

1

```
Ldi r16,0
Loop:
        Inc r16
        out PORTB,r16
        cpi r16,10
        Brne loop
```

2

Loop pertama: menaikkan pencacah setelah penulisan nilai ke PORTB. Nilai 0,1,2,....,9

Loop kedua: menaikkan pencacah sebelumpenulisan nilai ke PORTB. Nilai 1,2,....,10

CONTOH

Tuliskan kode assembly untuk mengirimkan angka 0 sampai dengan angka 99 ke PORTA. Gunakan struktur FOR..DO.

```
.Include “m32def.inc”
.org 0x0000
Rjmp utama
Utama:
Ldi r16,low(ramend)
Mov spl,r16
Ldi r16,high(ramend)
Mov sph,r16

Ldi r16,0xff ;port a output
Out DDRA,r16
```

```
Ldi r24,0

FOR:
  Out PORTA,r24
  Inc r24
  Cpi r24,99
  Brne FOR

End_for: rjmp End_for
```

PENGULANGAN WHILE...DO

- Merupakan variasi dari pengulangan FOR.
- Pengulangan ini memeriksa jika suatu hasil pengujian tertentu adalah benar dan melakukan instruksi-instruksi loop.

```
WHILE kondisi DO
        statement
END_WHILE
```

CONTOH

```
While1:
   In r16,PIND              ;baca pinD
   Cpi r16,1                ;bandingkan dengan 1
   Brne while1 _end         ; jika tidak sama akhiri loop
   Rcall port_adalah_1
   Rjmp while_1                  ;ulangi loop
While1_end: rjmp While1_end
```

PENGULANGAN REPEAT ... UNTIL

- Pengulangan ini sedikitnya mengeksekusi instruksi loop sebanyak satu kali.

```
REPEAT
        statement
UNTIL condition
```

- Dalam repeat until, statement dieksekusi kemudian conditon diperiksa. Jika benar maka loop diakhiri.

CONTOH

```
Repeat1:
    Rcall port_adalah_1      ; panggil port_adalah_1
    In r16,PIND              ;baca nilai PIND
    Cpi r16,1                ;bandingkan dengan angka 1
    Breq repeat1             ; jika benar lompat ke Repeat1
Akhir: rjmp Akhir
```

STRUKTUR CASE

- Merupakan struktur **multiway branch** yang melakukan pengujian pada register/variabel atau ekspresi nilai-nilai khusus atau rentang suatu nilai.

CASE expression
    Value_1: statement1
    Value_2: statement2
    .....
    Value_n: statement2
END_CASE

STRUKTUR CASE

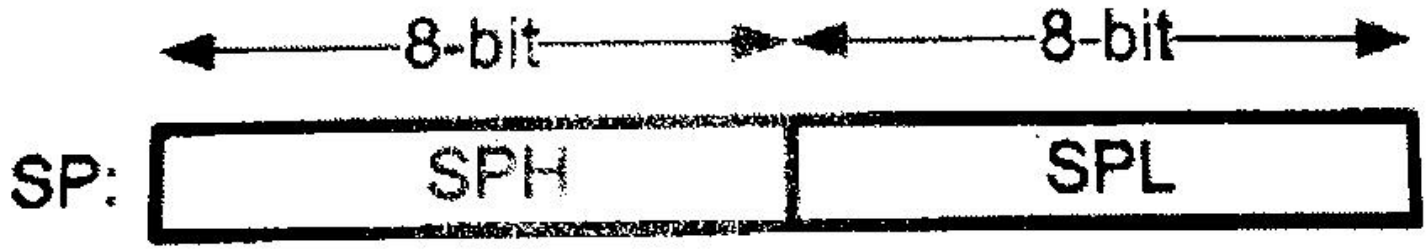
- Struktur case ini efektif untuk bermacam-macam kondisi yang harus dipilih. Misal ketika menerima nilai via UART. **CONTOH:**

```
In r16,UDR
Case_saya:
    Cpi r16,0
    Breq case_0
    Cpi r16,1
    Breq case_1
    Cpi r16,2
    Breq case_2
    Default: ;default code()
```

OPERASI STACK

- Merupakan struktur data satu dimensi.
- Suatu item ditambahkan dan dihilangkan dari ujung struktur **LAST IN FIRST OUT (LIFO).**
- Pada ruang memori I/O ada dua register untuk operasi stack register SPL (byte bawah register SP) dan register SPH (byte atas register SP).

Register Stack Pointer (SP)



INISIALISASI STACK POINTER

- Awal mikrokontroler ON SP = 0 yang merupakan alamat R0
- Kita harus menentukan inisialisasi SP pada awal program supaya menunjuk kemana saja didalam SRAM internal.
- Pada mikrokontroller AVR stack bergerak dari alamat tinggi ke alamat rendah pada saat PUSH maka terjadi proses decrement SP.
- Umumnya dilakukan inisialisasi SP pada lokasi memori tertinggi.

INISIALISASI STACK POINTER

- Pada assembler AVR **RAMEND** merepresentasikan alamat SRAM yang terakhir.
- SP terbagi dua SPH dan SPL
- Inisialisasi SP ke alamat terakhir Kita dapat me-load High byte RAMEND ke SPH dan low byte RAMEND ke SPL.

```
LDI r16,low(ramend)     ;r16=0x5F
OUT SPL,r16             ;SPL=0x5F
LDI r16,high(ramend)    ;r16=0x02
OUT SPH,r16             ;SPH=0x02
```

INSTRUKSI PUSH DAN POP

- Untuk menyalin sebuah byte dari register ke alamat puncak STACK digunakan opcode **PUSH**

Sintaks:

PUSH operand_sumber ; operand sumber R0..R31

- Eksekusi PUSH:
  - Pertama kali byte/data pada register disalin ke alamat yang ditunjuk oleh SP. Operand sumber (isi register) tidak berubah setelah byte di PUSH.
  - Setelah itu SP otomatis decrement (SP=SP-1)

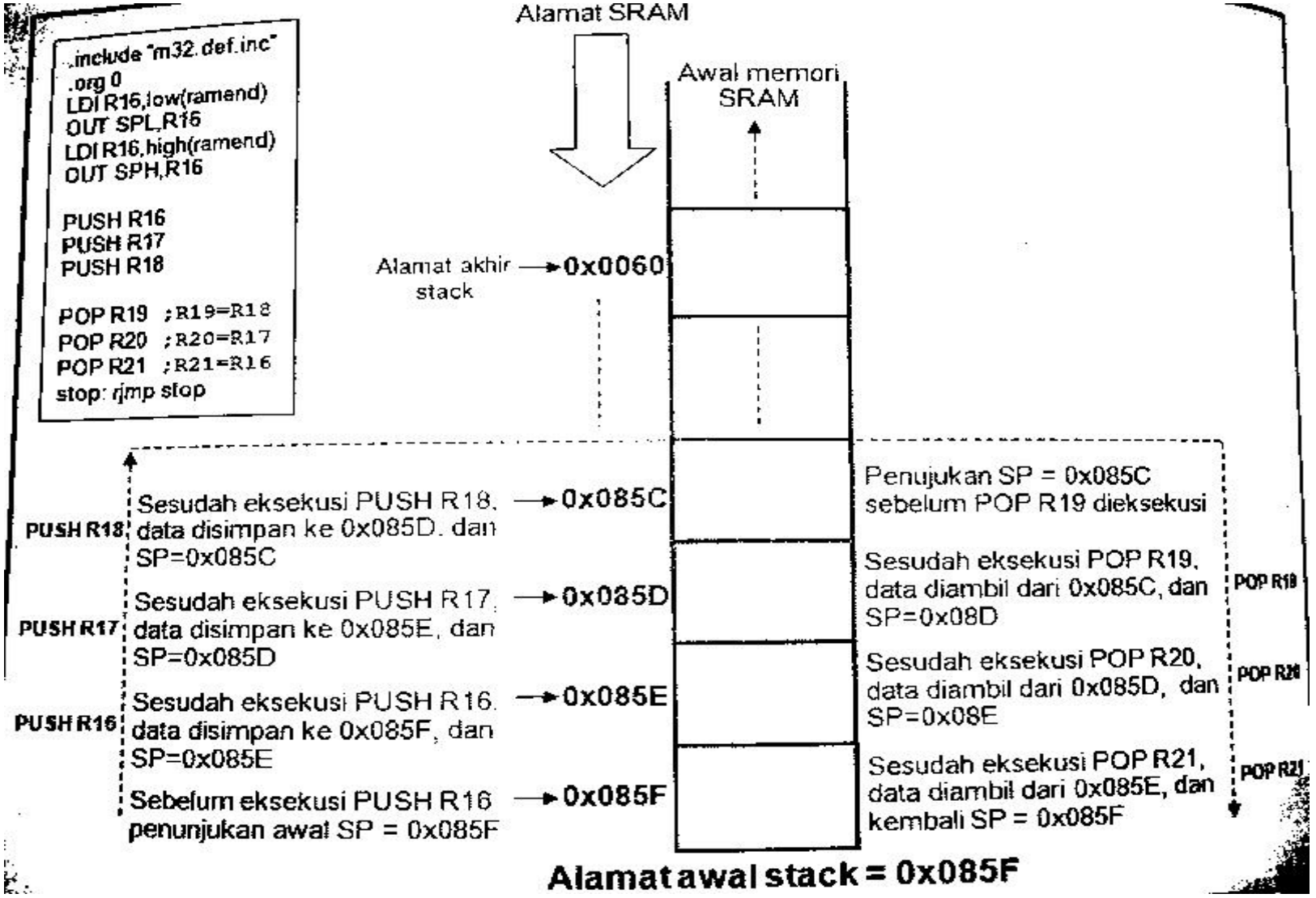
INSTRUKSI PUSH DAN POP

- Untuk menyalin byte/data dari puncak Stack ke dalam register digunakan opcode **POP**.
- **Sintaks:**

POP operand_sumber ; operand sumber R0..R31

- **Eksekusi POP:**
  - Pertama kali nilai SP secara otomatis increment (SP=SP+1).
  - Kemudian byte/data pada puncak stack yang alamatnya ditunjuk oleh nilai SP yang baru saja di increment disalin ke dalam operand_sumber (register).

INSTRUKSI PUSH DAN POP

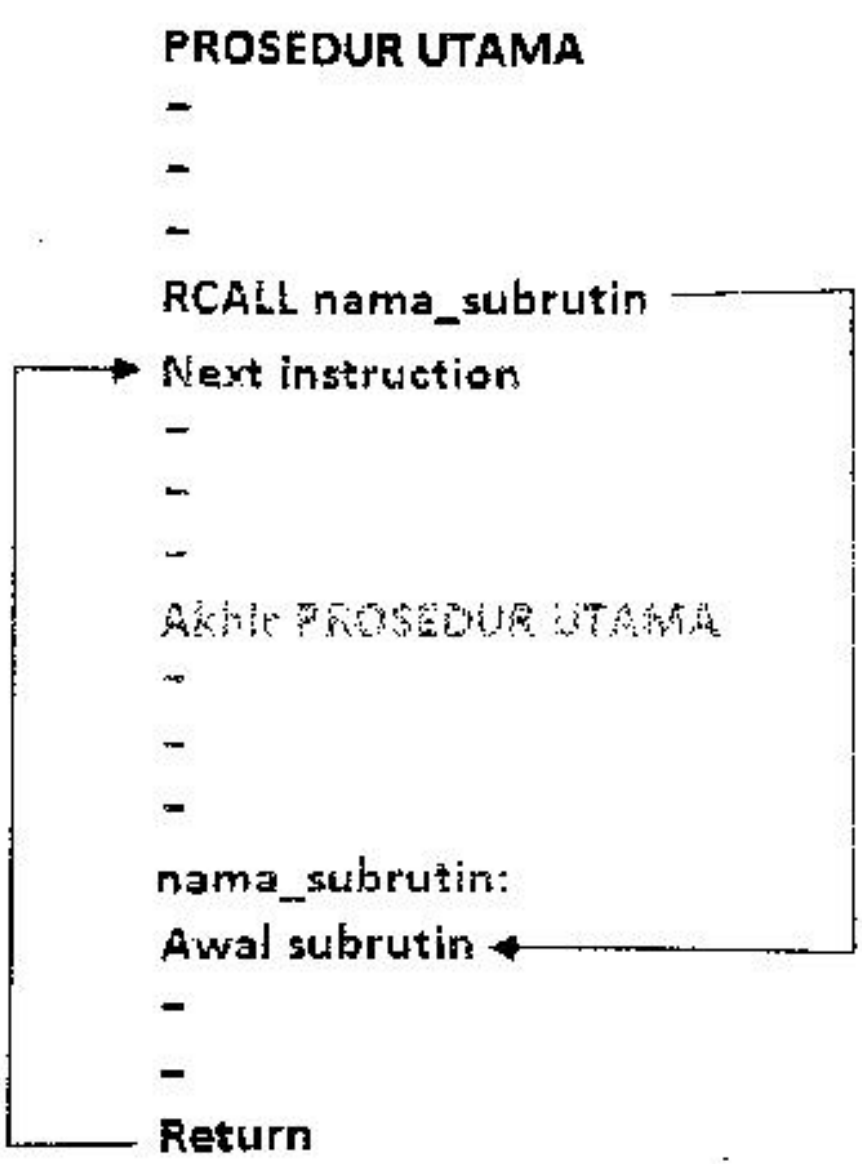


OPERASI SUBRUTIN (Prosedur)

- Salah satu prosedur/rutin dijadikan sebagai **main procedure (**prosedur utama**)** dan berisi entry point dalam program.
- Untuk melakukan suatu pekerjaan, main procedure akan memanggil salah satu dari subrutin-subrutin yang ada.
- Subrutin mungkin dapat memanggil subrutin lain atau memanggil dirinya sendiri.

OPERASI SUBRUTIN (Prosedur)

- Jika satu subrutin memanggil subrutin lain, maka control dialihkan ke subrutin yang dipanggil dan instruksinya dieksekusi.
- Pemanggilan subrutin umumnya mengembalikan control ke pemanggil pada instruksi berikutnya setelah pernyataan **CALL**.


PROSEDUR UTAMA
RCALL nama_subrutin
Next instruction
Akhir PROSEDUR UTAMA
nama_subrutin:
Awal subrutin
Return

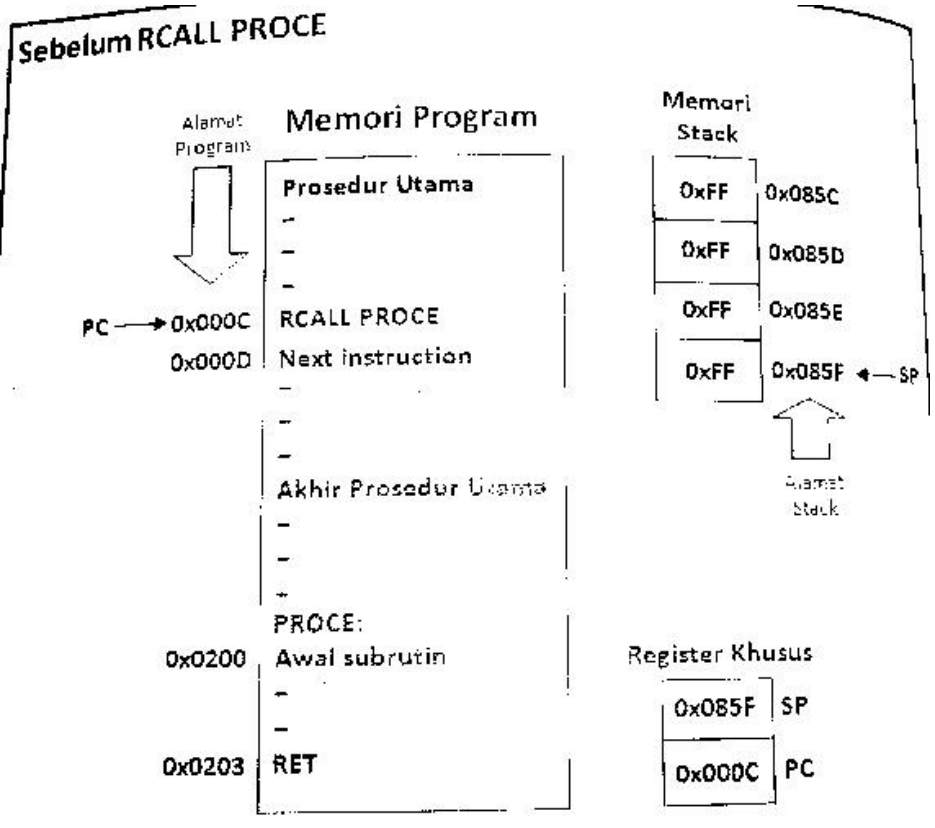
RCALL DAN RET

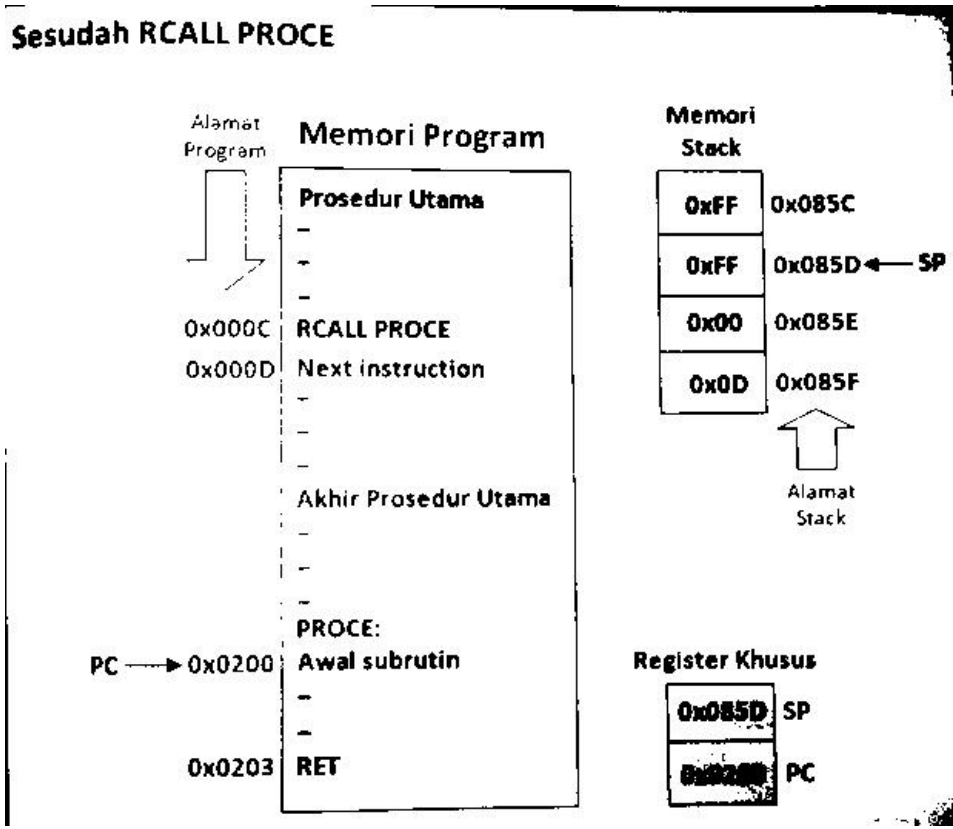
- Untuk memanggil subrutin digunakan instruksi **RCALL.**
- **Sintaks:**

RCALL nama_subrutin

- Eksekusi RCALL:
  - Alamat kembali (return address) ke program pemanggil secara otomatis disimpan kedalam stack. (alamat ini adalah alamat instruksi berikutnya yang berada tepat dibawah instruksi RCALL).
  - Program Counter menerima alamat instruksi dari subrutin. Alamat ini mentransfer control ke subrutin.

ILUSTRASI

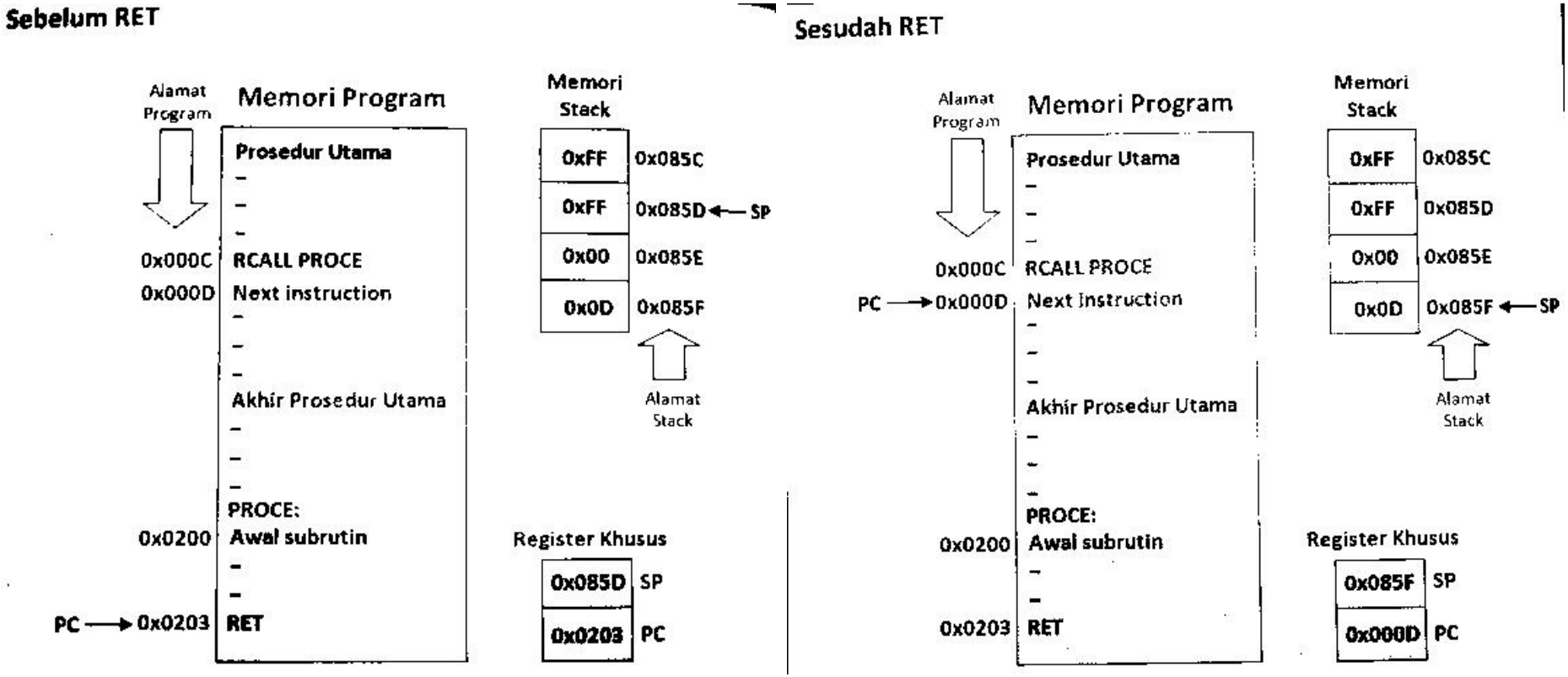

Sebelum RCALL PROCE
Alamat Program
Memori Program
Prosedur Utama
PC → 0x000C RCALL PROCE
0x000D Next instruction
Akhir Prosedur Utama
PROCE:
0x0200 Awal subrutin
0x0203 RET
Memori Stack
0xFF 0x085C
0xFF 0x085D
0xFF 0x085E
0xFF 0x085F ← SP
Alamat Stack
Register Khusus
0x085F SP
0x000C PC


Sesudah RCALL PROCE
Alamat Program
Memori Program
Prosedur Utama
0x000C RCALL PROCE
0x000D Next instruction
Akhir Prosedur Utama
PROCE:
PC → 0x0200 Awal subrutin
0x0203 RET
Memori Stack
0xFF 0x085C
0xFF 0x085D ← SP
0x00 0x085E
0x0D 0x085F
Alamat Stack
Register Khusus
0x085D SP
PC

RCALL DAN RET

- Untuk kembali dari subrutin ke program pemanggil digunakan instruksi **RET.**
- Eksekusi RET menyebabkan isi stack disalin kembali kedalam PC. PC sekarang berisi alamat kembali dan control program akan kembali ke program pemanggil.

ILUSTRASI


Sebelum RET
Alamat Program
Memori Program
Prosedur Utama
0x000C RCALL PROCE
0x000D Next instruction
Akhir Prosedur Utama
PROCE:
0x0200 Awal subrutin
PC → 0x0203 RET
Memori Stack
0xFF 0x085C
0xFF 0x085D ← SP
0x00 0x085E
0x0D 0x085F
Alamat Stack
Register Khusus
0x085D SP
0x0203 PC
Sesudah RET
Alamat Program
Memori Program
Prosedur Utama
0x000C RCALL PROCE
PC → 0x000D Next Instruction
Akhir Prosedur Utama
PROCE:
0x0200 Awal subrutin
0x0203 RET
Memori Stack
0xFF 0x085C
0xFF 0x085D
0x00 0x085E
0x0D 0x085F ← SP
Alamat Stack
Register Khusus
0x085F SP
0x000D PC

CONTOH DALAM PROGRAM

```
; Contoh Program Penerapan Prosedur
.include "m32def.inc"

.org 0
ldi r16,low(RAMEND)

out SPL,r16
ldi r16,high(RAMEND)

out SPH, r16
  ldi r16,0x66
RCALL PROCE                    ;panggil subrutin PROCE

back:
  ldi r17,0x77
  ldi r18,0x88
  ldi r19,0x99
  ldi r20,0x00
stop: rjmp stop

PROCE:
  ldi r21,0x11                 ;awal subrutin PROCE
  ldi r22,0x22
RET                            ;kembali ke program utama dengan label back
```

MENGHITUNG WAKTU TUNDA (DELAY TIME)

- 2 hal yang perlu diperhatikan untuk menghitung waktu tunda (delay time) didalam program bahasa assembly, yaitu:
  1. Frekuensi Kristal mikrokontroler yang digunakan
  2. Siklus clock pada setiap instruksi yang digunakan dalam proses loop.

CONTOH

Wait:

ldi r16,_____

Delay:

dec r16

brne delay

$$T_{delay}=(N_{mc}*L_{cnt}-1)t_{mc}$$

T_{delay} lama delay yang akan digenerate pada loop

t_{mc} periode siklus mesin ($1/F_{clk}$)

N_{mc} jumlah siklus mesin dalam 1 loop

L_{cnt} jumlah perulangan untuk loop

Misal L_{cnt} = 255 dan F_{clk}=16MHz

$T_{delay}=(N_{mc}*L_{cnt}-1)t_{mc}$

$T_{delay}=(3*255-1)(0,0625us)$

$T_{delay}=(3*255-1)(0,0625us)$

$T_{delay}= 47,75us$

CONTOH

Wait:

```
ldi r16,_____
```

Delay:

```
nop
dec r16
brne delay
```

Misal L_{cnt} = 255 dan F_{clk}=16MHz

$T_{delay}=(N_{mc}*L_{cnt}-1)t_{mc}$

$T_{delay}=(4*255-1)(0,0625us)$

$T_{delay}=(4*255-1)(0,0625us)$

$T_{delay}= 63,6875us$

- Untuk menghasilkan delay 500us, misal kita akan menginisialisasikan delay selama 50us untuk r16 kemudian menulis loop luar yang akan menjalankan loop bagian dalam selama 10 kali, sehingga total delay mencapai 500us

- $T_{delay}=(N_{mc}*L_{cnt}-1)t_{mc}$
- $L_{cnt}=(T_{delay}/t_{mc}+1)N_{mc}$
- $L_{cnt}=(T_{delay}*F_{clk}+1)N_{mc}$

- Set Lcnt untuk delay 50us

  Lcnt = (50µs/0.0625µs+ 1)/4 ≈ 200 = 0xC8

```
wait:
        ldi r16, 0xC8           // 200
delay:
        nop                     // 1
        dec r16                 // 1 clock cycle
        brne delay              // 2/ 1
```

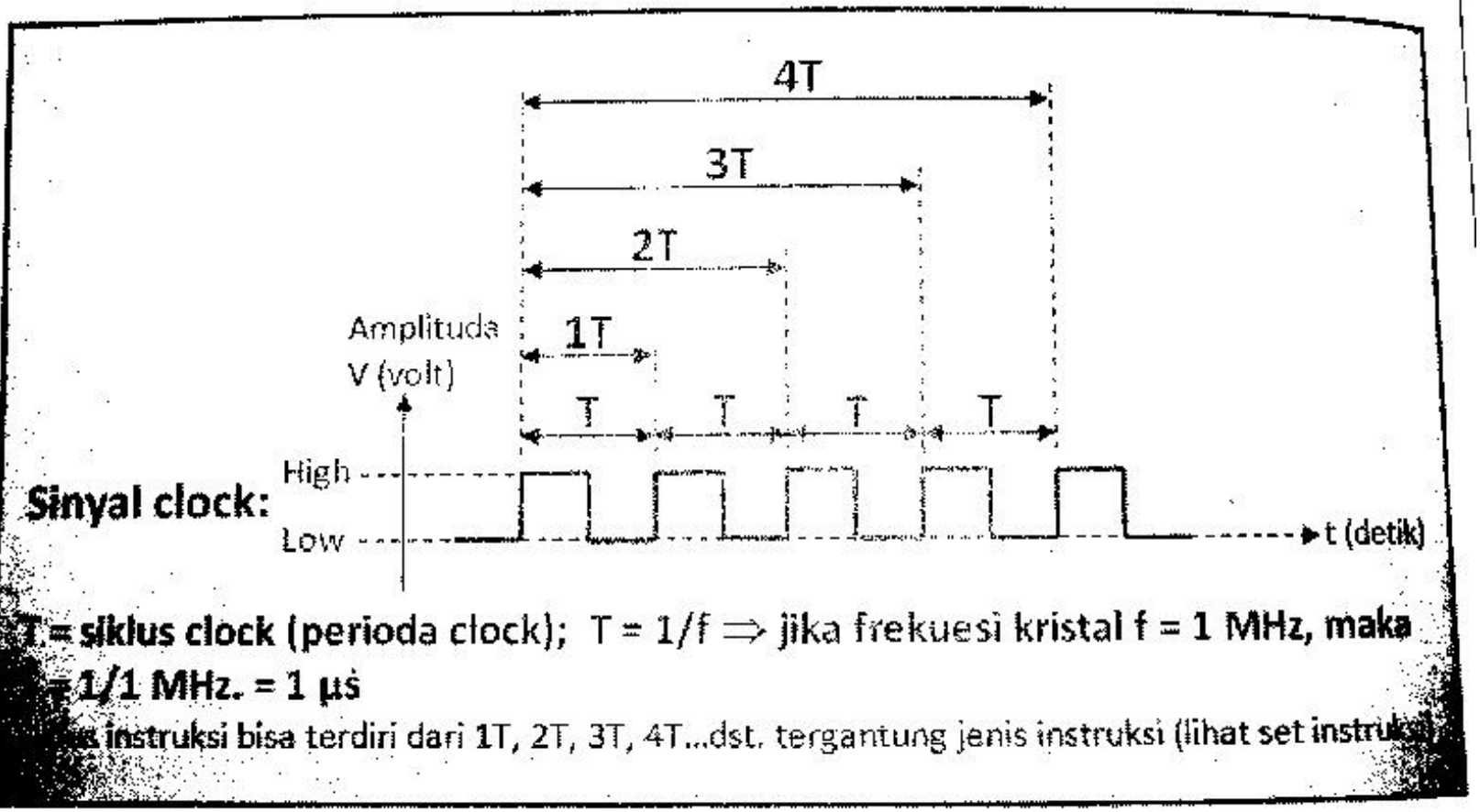
Latihan:

Buat Loop bagian luar

Hitunglah waktu tunda yang ditimbulkan oleh program berikut jika digunakan frekuensi Kristal 1 MHz.

KODE INSTRUKSI	KETERANGAN
.include "m32def.inc"	
.org 0x0000	
ldi r16, low(RAMEND)	
out SPL, r16	
ldi r16, high(RAMEND)	
out SPH, r16	
LAGI:	
LDI R16,0XFF	
OUT DDRA,R16	
LDI R16,0XAA	
OUT PORTA,R16	
CALL TUNDA	;eksekusi waktu tunda
LDI R16,0X55	
OUT PORTA,R16	
rjmp LAGI	
	;program TUNDA diletakkan pada alamat 0x200
.org 0x200	
TUNDA: LDI r16,250	;inisialisasi nilai pengulangan (counter)
LUP: NOP	;no operation
DEC r16	;decrement r16
BRNE LUP	;selama r16 ≠ 0, maka lompat ke label LUP
	;kembali ke baris tepat di bawah pemanggil subrutin
RET	

Penjelasan perhitungan waktu tunda dapat mengacu pada gambar berikut.



Sinyal clock:

T = siklus clock (perioda clock); T = 1/f ⇒ jika frekuesi kristal f = 1 MHz, maka T = 1/1 MHz. = 1 µs

Satu instruksi bisa terdiri dari 1T, 2T, 3T, 4T...dst. tergantung jenis instruksi (lihat set instruksi)

```
TUNDA: LDI r16,250          ;1 siklus clock
LUP:   NOP                  ;1 siklus clock
DEC r16                     ;1 siklus clock
                            ;2/1 siklus clock (lompat = 2, tidak
BRNE LUP                    lompat = 1)
                            ;4 siklus clock
RET
```

Dari program terlihat bahwa yang menjadi perhitungan waktu tunda adalah instruksi yang berada di dalam procedure TUNDA, sehingga total Waktu Tunda = [1 + (1 + 1 + 2) x 250 + 4] x 1 µs = 1005 µs.

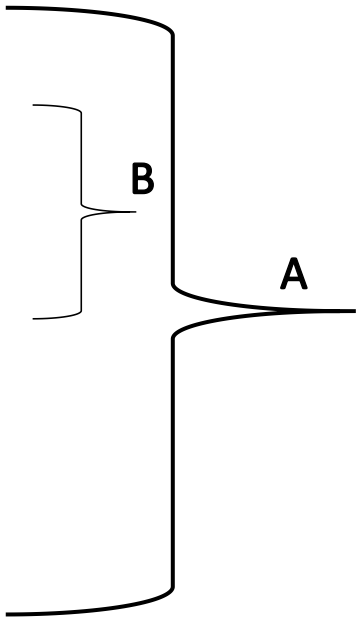
Perlu diperhatikan kembali bahwa instruksi BRNE LUP dieksekusi sebanyak 250 kali yang terdiri dari: 249 kali melompat ke label LUP dan 1 kali tidak melompat. Artinya total waktu tunda hasil perhitungan di atas harus dikurangi 1 siklus *clock*, sehingga waktu tunda yang sebenarnya = 1005 µs – 1 µs = 1004 µs = 1,004 ms.

TUNDA MENGGUNAKAN LOOP BERSARANG

- Merupakan sekumpulan instruksi/operasi yang dilakukan berulang **n kali** didalam satu loop, kemudian loop tersebut diulang kembali sebanyak **m kali** Total perulangan = n*m.

- **Misal kristal = 4MHz $T=1/F_{clk}$ = 0,250us**

```
        Ldi r16,5       ; 1
lagi:   Ldi r17,4
lagi1:  dec r17
        nop
        brne lagi1
        dec r16
        nop
        brne lagi
        nop             ;1
        nop             ;1
```



Total cycle = 3 +A

A= 5*(1+B+1+1+2)-1
A= (5B+24)cycle

B=4*(1+1+2)-1
B=15cycle

A=(5(15)+24)cycle
A=99cycle

Total = 3+A - 3+99 = 102cycle
Waktu eksekusi = 102*0,250
Waktu ekseskusi= 25,25us

LATIHAN

1. Hitung waktu tunda untuk subrutin dibawah ini dimana Kristal = 4MHZ

```
LDI R16,250
LOOP1: LDI R17,4
LOOP2: NOP
       DEC R17
       BRNE LOOP2
       DEC R16
       BRNE LOOP1
```

2. Hitung waktu tunda untuk subrutin dibawah ini dimana Kristal = 8MHZ

```
LDI R16,25
LOOP1:  LDI R17,8
LOOP2:  NOP
        NOP
        NOP
        DEC R17
        BRNE LOOP2
        DEC R16
        BRNE LOOP1
```