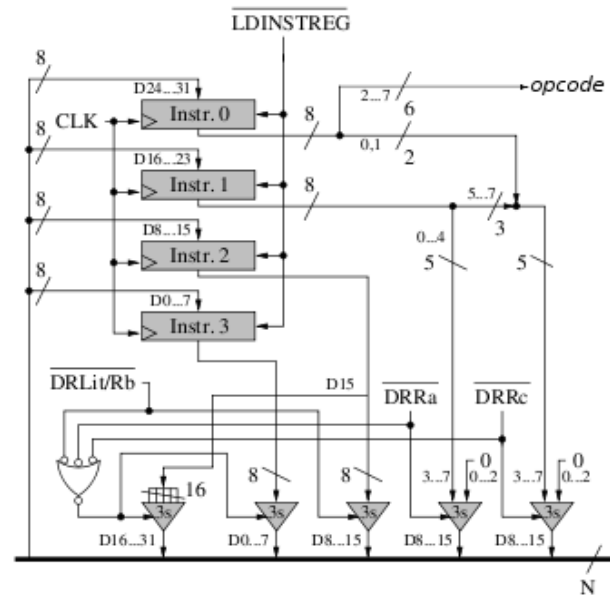
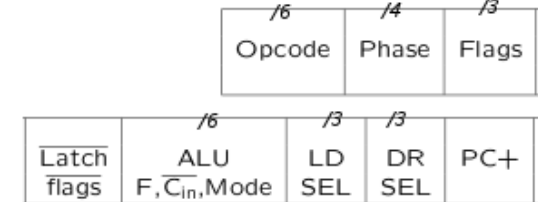
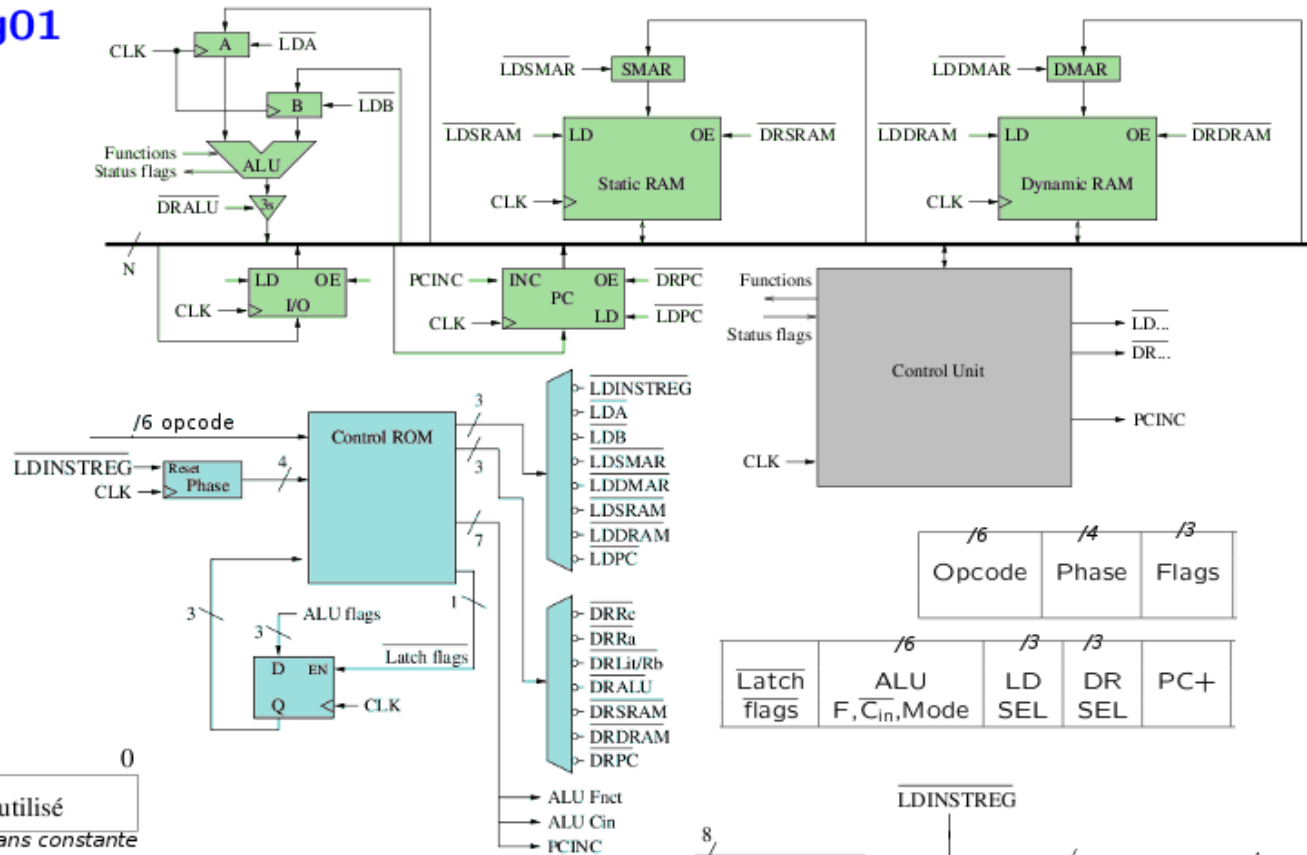
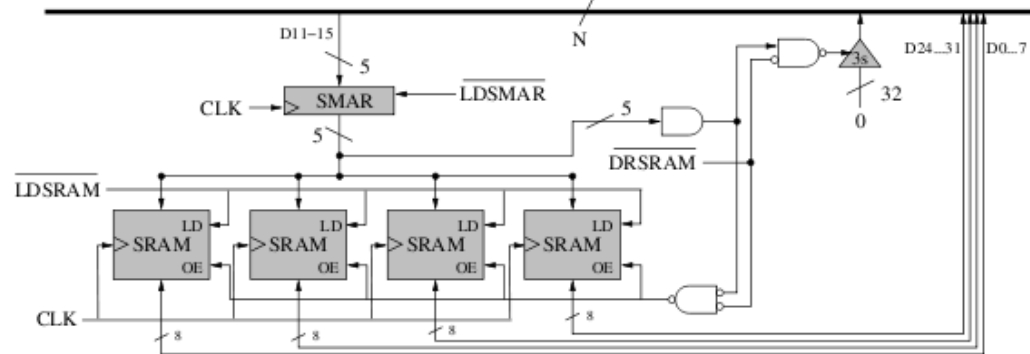
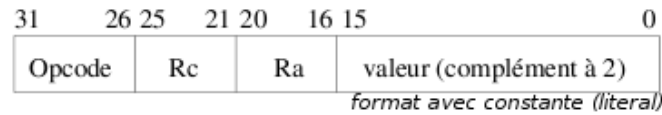
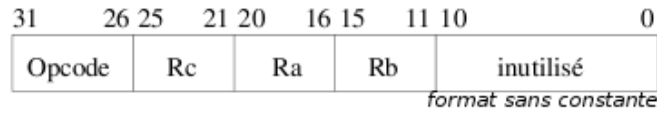


Computation structures
Tutorial 2: μ -code for ULg01
and β -assembly

ULg01

Control Inputs							Output
F3	F2	F1	F0	C	M		
0	0	1	1	1	1	0	0
1	1	0	0	1	1	0	0xffffffff
1	1	1	1	1	1	1	A
1	0	1	0	1	1	1	B
1	1	1	1	1	0	0	A-1
0	0	0	0	0	0	0	A+1
1	0	0	1	1	0	0	A+B
0	1	1	0	0	0	0	A-B
1	0	1	1	1	1	1	A and B
1	1	1	0	1	1	1	A or B
0	1	1	0	1	1	1	A xor B
0	0	0	0	1	1	1	not A
1	1	0	0	1	0	0	A+A
0	1	1	0	1	0	0	A-B-1
1	0	0	1	x	1	1	not (A xor B)
0	1	0	0	x	1	1	not (A and B)

flags:
 $E=1 \iff \text{Alu out} = 0xffffffff$
 C : negated carry out
 N = most sign. Alu out: a31



β -assembly

- β -assembly is a low-level human-readable language than can be used to program the β -machine
- Processed by an **assembler** to produce machine code (e.g. instructions to be loaded into INSTREG)
- The input of the assembler is a sequence of constant expressions
- The language provides some **mechanisms** for facilitating writing sequences of constant expressions

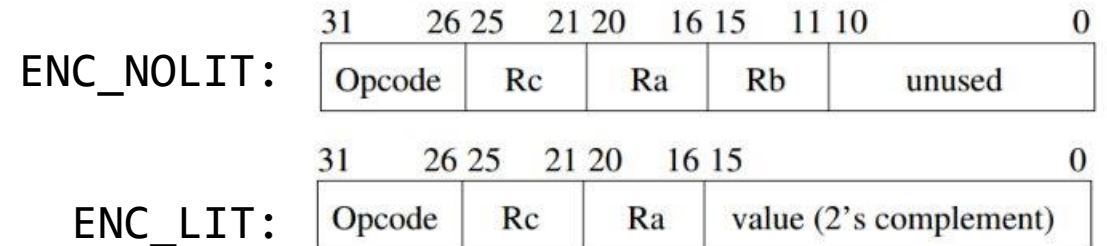
β -assembly – Identifiers, macros and comments

- You can define identifiers: `X = 0xF20`
- « . » is a special identifier containing the number of the **next byte to be added in the sequence**
- A position can be assigned to an identifier: `id: |`; equivalent to `id = .`
- Macros are parametrizable program fragment that can be reused
- They can be defined as: `.macro name(p1, ..., pn) body`
- And used as: `name(v1, ..., vn)`
- You can comment a line by starting it with « | » or « | ; »

β -assembly – ULg01

β -assembly for ULg01 is built using those mechanisms:

```
.macro WORD(x) x%0x100 (x>>8)%0x100  
.macro LONG(x) WORD(x) WORD(x >> 16)
```



```
.macro ENC_NOLIT(OP, RA, RB, RC) LONG((OP<<26)+((RC%0x20)<<21)+((RA%0x20)<<16)+((RB%0x20)<<11))  
.macro ENC_LIT(OP, RA, CC, RC) LONG((OP<<26)+((RC%0x20)<<21)+((RA%0x20)<<16)+(CC%0x10000))  
.macro ENC_ADRLIT(OP, RA, RC, label) ENC_LIT(OP, RA, RC, (label - (. + 4)) >> 2 )
```

```
;; Registers  
r0 = 0b0  
R0 = r0  
;; ...  
r31 = 0b11111  
R31 = r31
```

β -assembly – ULg01

Without a literal		With a literal	
Opcode	name	Opcode	name
0x20	ADD	0x30	ADDC
0x21	SUB	0x31	SUBC
0x22	MUL	0x32	MULC
0x23	DIV	0x33	DIVC

Example of instruction definition:

Opcode	name
0x1D	BEQ/BF
0x1E	BNE/BT

```
|; ADD(Ra, Rb, Rc)      Reg[Rc] <- Reg[Ra] + Reg[Rb]
```

```
.macro ADD(Ra,Rb,Rc)    ENC_NOLIT(0x20,Ra,Rb,Rc)
```

```
|; BEQ(Ra, label, Rc)  Reg[Rc] <- PC; if Reg[Ra] == 0 then PC <- Mem[PC + CC]
```

```
.macro BEQ(Ra,label,Rc) ENC_ADRLIT(0x1D,Ra,Rc,label)
```

```
|; MOVE(Ra, Rb)        Reg[Rb] <- Reg[Ra]
```

```
.macro MOVE(Ra,Rb)     ADD(Ra,R31,Rb)
```

β -assembly – Writing a program

As simple as defining identifiers and calling macros:

```
.include beta.uasm |; Include the definition of beta-assembly
```

```
main:
```

```
CMOVE(0x25, R1) |; Reg[R1] <- 0x25  
CMOVE(0x876, R2) |; Reg[R2] <- 0x876  
ADD(R1, R2, R3) |; Reg[R3] <- Reg[R1] + Reg[R2]  
MULC(R3, 4, R3) |; Reg[R3] <- Reg[R3] * 4  
LD(R3, 0, R4) |; Reg[R4] <- Mem[Reg[R3] + 0]
```

β -assembly – Functions

For supporting function/procedure calls we need:

- Allowing procedure to return a value



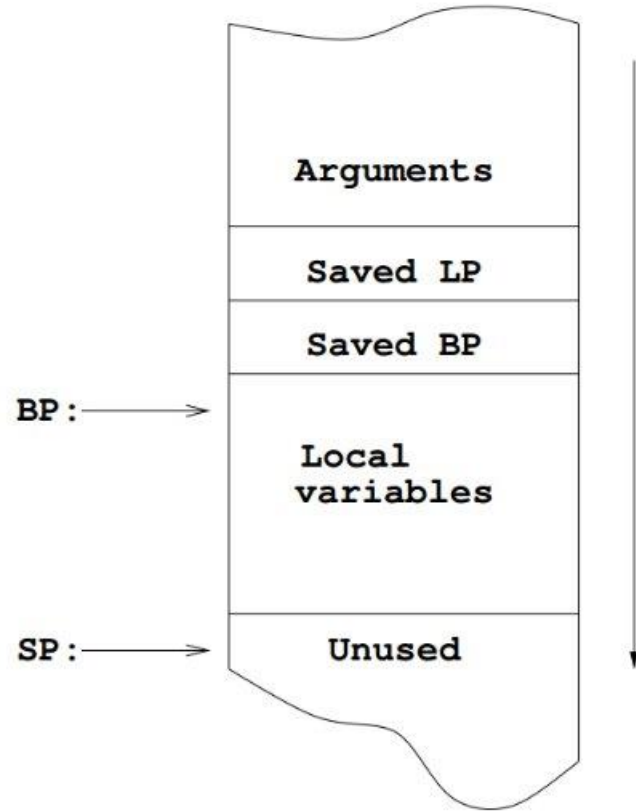
Use R0 to store returned value

- Passing arguments to the procedures
- Allowing procedures to call other procedures, including itself
- Making it possible for a procedure to use local variables



Use a **stack** !

β -assembly – Stack



Special registers:

- **BP (r27)**: base of frame pointer
- **SP (r28)**: stack pointer
- **LP (r29)**: linkage pointer

The **frame** is the stack area used by a procedure.

LP contains the address of the instruction to branch to when the execution of a procedure is over.

SP contains the address of the first free element on top of the stack

β -assembly – How to call a procedure

In the calling procedure:

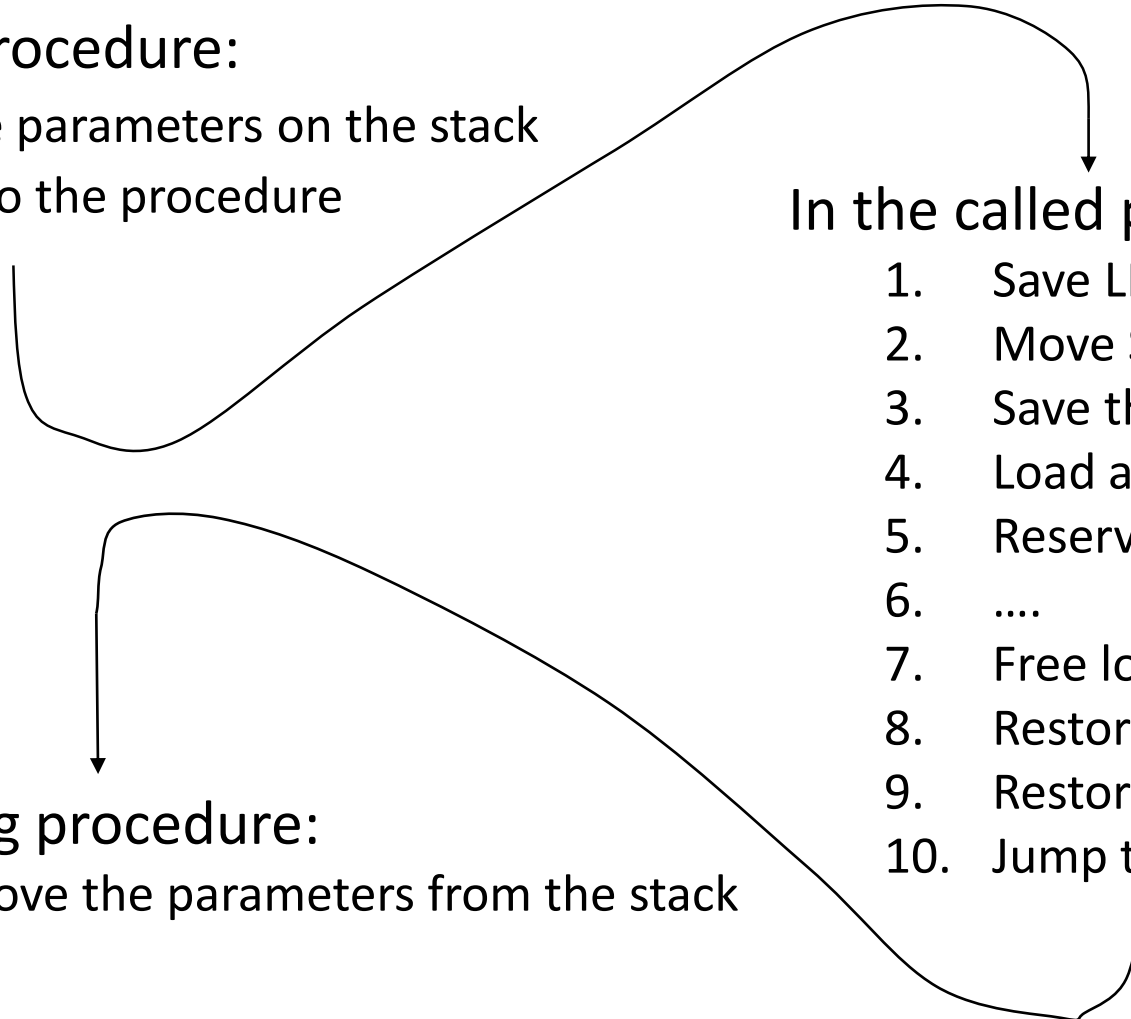
1. Push the parameters on the stack
2. Branch to the procedure

In the called procedure:

1. Save LP and BP on the stack
2. Move SP in BP
3. Save the registers you will use on the stack
4. Load arguments (if any)
5. Reserve space for local variables (if any)
6.
7. Free local variable space (if any)
8. Restore saved registers
9. Restore BP and LP
10. Jump to LP

In the calling procedure:

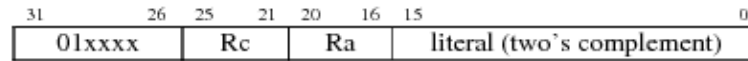
1. Remove the parameters from the stack



Comment programmer ULg01 ?

<http://6004.csail.mit.edu/Fall00/handouts/beta.ps>

<http://www.run.montefiore.ulg.ac.be/~martin/SDO/>



Identificateur = expression

```

X: . = .+4
.macro WORD(x) (x)%256 (x)/256

```

```

LD:      Reg[Rc] ← Mem[Reg[Ra] + SEXT(literal)]
ST(Rc,lit,Ra) Mem[Reg[Ra] + SEXT(literal)] ← Reg[Rc]
JMP:     Reg[Rc] ← PC + 4; PC ← Reg[Ra]
BEQ:     Reg[Rc] ← PC + 4; if Reg[Ra] = 0 then PC ← PC + 4 + 4*SEXT(literal)
BNE:     Reg[Rc] ← PC + 4; if Reg[Ra] ≠ 0 then PC ← PC + 4 + 4*SEXT(literal)
LDR:     Reg[Rc] ← Mem[PC + 4 + 4*SEXT(literal)]

```

SP=R29
LP=R28
BP=R27

PUSH(Rx)
POP(Rx)
ALLOCATE(n)
DEALLOCATE(n)

```

.include beta.uasm
    CMOVE(tos,SP)
main:
| Do your stuff here

.breakpoint
    HALT()

iam_a_var: LONG(27)
Y = 9

| Placeholder for your
| own functions.

LONG(0xdeadcafe)
tos:
    STORAGE(1024)
    | this room is for
    | the stack

| WORD(val)      | Assemble val as a 16-bit datum
| LONG(val)      | Assemble val as a 32-bit datum
| STORAGE(NWORDS) | Reserve NWORDS 32-bit words of DRAM

.macro BF(RA, LABEL, RC)      BEQ(RA, LABEL, RC)
.macro BT(RA, LABEL, RC)      BNE(RA, LABEL, RC)
.macro BR(LABEL, RC)          BEQ(r31, LABEL, RC)
.macro MOVE(RA, RC)           ADD(RA, R31, RC)
.macro CMOVE(CC, RC)          ADDC(R31, CC, RC)

.macro PUSH(RA)               ADDC(SP, 4, SP) ST(RA, -4, SP)
.macro POP(RA)                LD(SP, -4, RA)  ADDC(SP, -4, SP)

.macro CALL(label)           BR(label, LP)

.macro RTN()                 JMP(LP)
.macro XRTN()                JMP(XP)
.macro ALLOCATE(N)           ADDC(sp, N*4, sp)
.macro DEALLOCATE(N)        SUBC(sp, N*4, sp)

```

```

PC → 1FF8:00000000    HALT()
      1FFC:00000000    HALT()
      0000:C3BF0010    ADDC(R31, 16, SP)
      0004:00000000    main: HALT()
      0008:0000001B    iam_a_var: illop
      000C:DEADCAFE    illop
      0010:00000000    tos: HALT()
      0014:00000000    HALT()
      0018:00000000    HALT()
      001C:00000000    HALT()

```

	0	1	2	3	4	5	6	7
18	LD	ST		JMP		BEQ	BNE	LDR
20	ADD	SUB	MUL	DIV	CMPEQ	CMPLT	CMPLE	
28	AND	OR	XOR		SHL	SHR	SRA	
30	ADDc	SUBc	MULc	DIVc	CMPEQc	CMPLTc	CMPLEc	
38	ANDc	ORc	XORc		SHLc	SHRc	SRAc	

