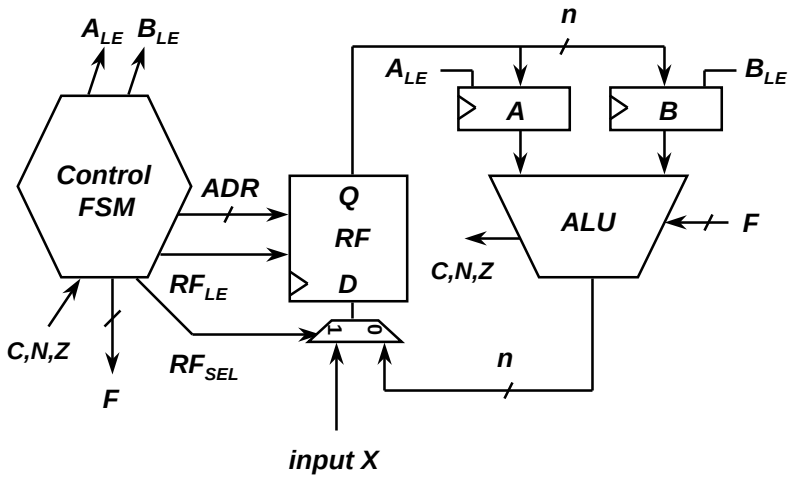
6004 Spring 1998: L20

Introduction to Machine Language (Controlling a Beast!)

1

Recap: Datapath



2

6004 Spring 1998: L20

Recap: Control FSM vs. Machine Language

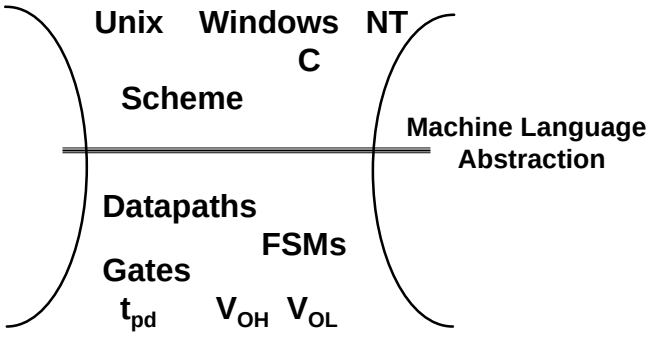
$$K = (C + D) * (C - D)$$

	RF_{SEL}	RF_{LE}	ADR	A_{LE}	B_{LE}	F	
s_1	1	1	000	0	0	–	$R_0 \leftarrow C$
s_2	1	1	001	0	0	–	$R_1 \leftarrow D$
s_3	0	0	000	1	0	–	$A \leftarrow \langle R_0 \rangle$
s_4	0	0	001	0	1	–	$B \leftarrow \langle R_1 \rangle$
s_5	0	1	000	0	0	A+B	$R_0 \leftarrow \langle A \rangle + \langle B \rangle$
s_6	0	1	001	0	0	A-B	$R_1 \leftarrow \langle A \rangle - \langle B \rangle$
s_7	0	0	000	1	0	–	$A \leftarrow \langle R_0 \rangle$
s_8	0	0	001	0	1	–	$B \leftarrow \langle R_1 \rangle$
s_9	0	1	000	0	0	A*B	$R_0 \leftarrow \langle A \rangle * \langle B \rangle$

3

Machine Language (ML)

A <u>contract</u> between software (compiler, programmer) and hardware (controller, datapath)



4

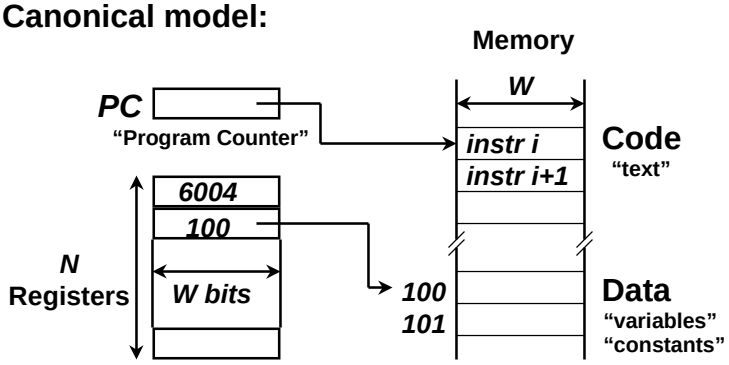
6004 Spring 1998: L20

ML a.k.a. Instruction Set Architecture

Two components:

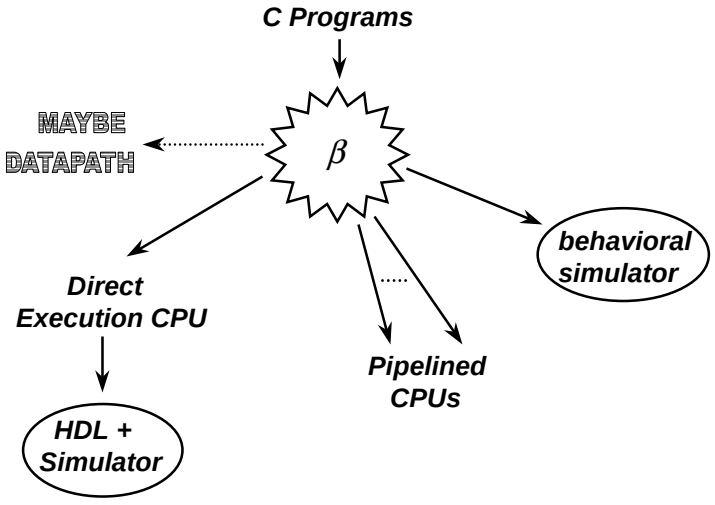
- Visible state
- Operations on state (instructions)

Canonical model:



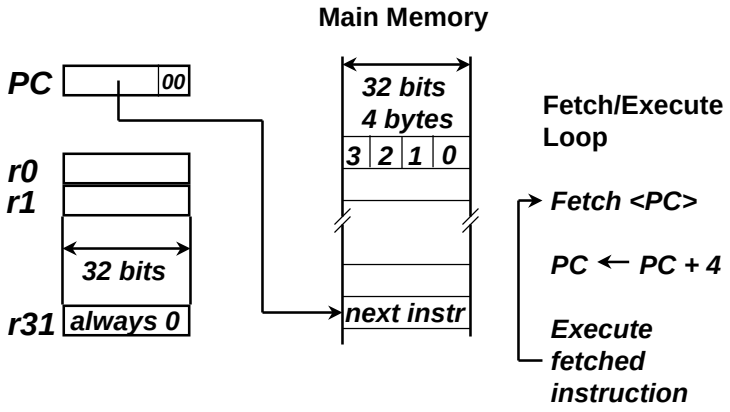
5

6.004 Processors and ISA's: Big Picture



6

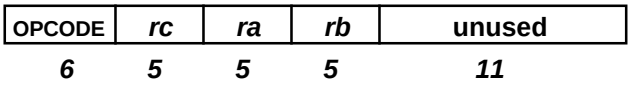
β Visible State and Computation Model



7

Binary Instruction Coding

What the β machine sees:



What we prefer to write:

symbolic machine language*

ADD(ra, rb, rc)

Register Transfer Notation

rc ← <ra> + <rb>

"Add the contents of *ra* to the contents of *rb* and store the result in *rc*".

*** also called symbolic assembly language**

8

β Instruction Formats

Form 1:

OPCODE	*rc*	*ra*	*rb*	unused
6	*5*	*5*	*5*	*11*

ALU operations with two source registers

Form 2:

OPCODE	*rc*	*ra*	literal C (signed)
6	*5*	*5*	*16*

ALU operations with one source register and constant
Loads and stores
Branches and Jumps

100000	00100	00010	00011	00000000000

written as ADD(_____)

9

Arithmetic Operations

ADD(ra, rb, rc) ***rc ← <ra> + <rb>***

"Add the contents of *ra* to the contents of *rb* and store the result in *rc*".

ADDC(ra, const, rc) ***rc ← <ra> + const***

"Add the contents of *ra* to *const* and store the result in *rc*".

16-bit signed constant

Similarly for:
SUB
MUL
AND
OR
XOR
SHL
SHR

10

Loads and Stores

LD(ra, C, rc) ***address ← <ra> + C***
rc ← Memory[address]

"Fetch into *rc* to the contents of the memory location whose address is *C* plus the contents of *ra*".

C is sign extended to 32 bits

ST(rc, C, ra) ***address ← <ra> + C***
Memory[address] ← <rc>

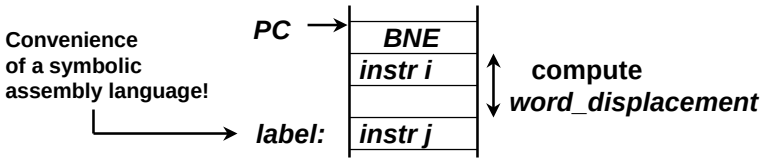
"Store the contents of *rc* into the memory location whose address is *C* plus the contents of *ra*".

11

Branch if nonzero

BNE(ra, label, rc) ***PC ← <PC> + 4***
rc ← <PC>
if <ra> nonzero then
PC ← <PC> +
4 word_displacement

"Fetch into *rc* the address of the next instruction. If ra does not contain zero then add *4* times *word_displacement* to the value of *PC*".



12

Branch if zero and Jump

BEQ(ra, label, rc) *PC ← <PC> + 4*
rc ← <PC>
if <ra> zero then
PC ← <PC> +
4 word_displacement

JMP(ra, rc) *PC ← <PC> + 4*
rc ← <PC>
PC ← <ra>

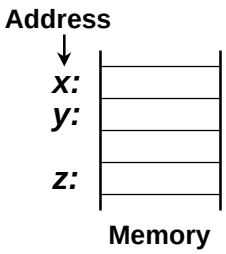
"Fetch into *rc* the address of the next instruction. Load the *PC* with the address contained in *ra*".

13

Let's Play "Compiler" !

Rules:

Variables live in memory
Operations done in ALU
Registers hold temporary values



Statement:

z = (x + y) * (x - y) ;

LD(r31, x, r1)
LD(r31, y, r2)
ADD(r1, r2, r3)
SUB(r1, r2, r4)

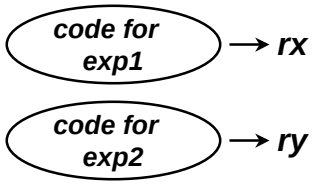
14

Compiling Statements

Statement:
variable = expression ;

ST(rz, var_addr, r31)

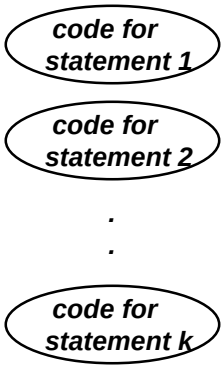
expression:
exp1 OP exp2



15

Compiling Blocks of Statements

Block:
{ statement 1 ;
statement 2 ;
.
.
statement k ; }

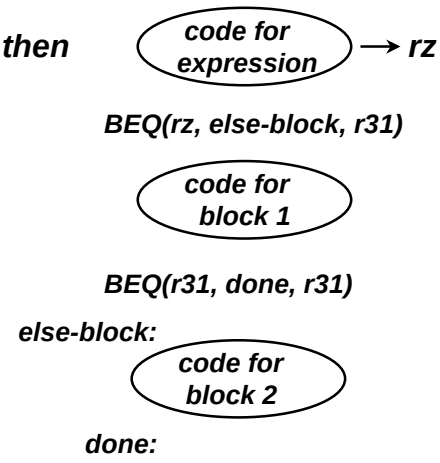


16

Conditional Statements

```
if expression then
    block 1
else
    block 2 ;
```

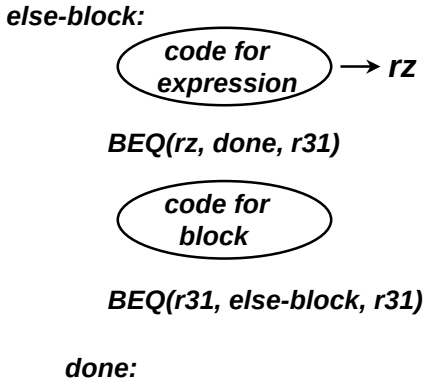
a.k.a. predicate
= 0 implies false
≠ 0 implies true



17

Iterations

```
while expression do
    block ;
```

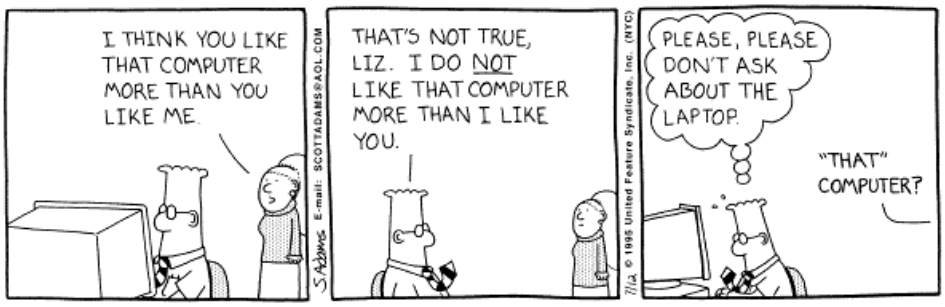


18

6004 Spring 1998: L20

Next Time: Programming Tools

Dilbert : S. Adams



19