How Computers Work

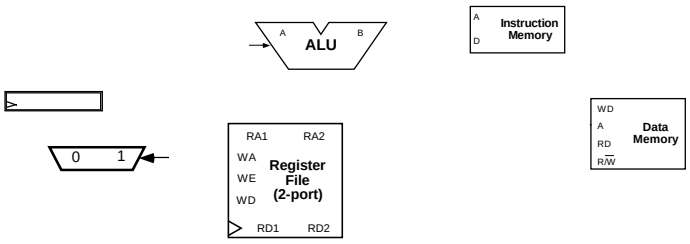
Lecture 3

A Direct Execution RISC Processor: The Unpipelined BETA

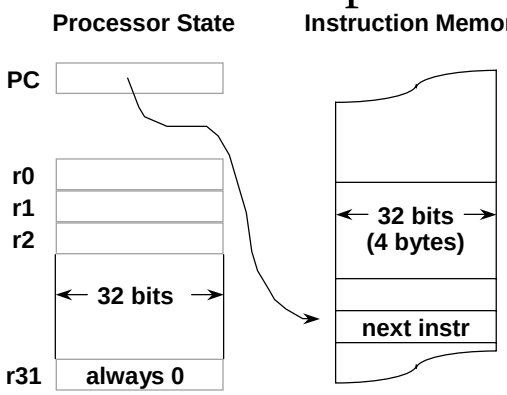
What you can do with very little:

Each instruction class can be implemented using a few simple components.

Components:



Review: β Model of Computation

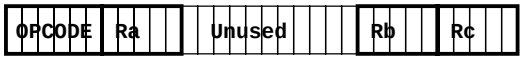


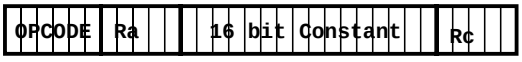
Fetch/Execute Loop:

- **Fetch <PC>**
- **PC ← *<pc> + 1***
- **Execute fetched instruction**
- **Repeat!**

Review: BETA Instructions

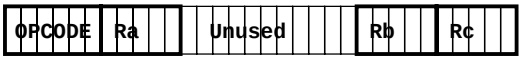
Two 32-bit Instruction Formats:





Review: β ALU Operations

What the machine sees (32-bit instruction word):

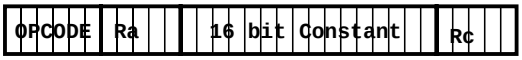


What *we* prefer to see: symbolic ASSEMBLY LANGUAGE

ADD(ra, rb, rc) **rc ← <ra> + <rb>**

"Add the contents of ra to the contents of rb; store the result in rc"

Alternative instruction format:



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

"Add the contents of ra to const; store the result in rc"

SIMILARLY FOR:

- **SUB, SUBC**
- **(optional)**
  MUL, MULC
  DIV, DIVC

BITWISE LOGIC:

- **AND, ANDC**
- **OR, ORC**
- **XOR, XORC**

SHIFTS:

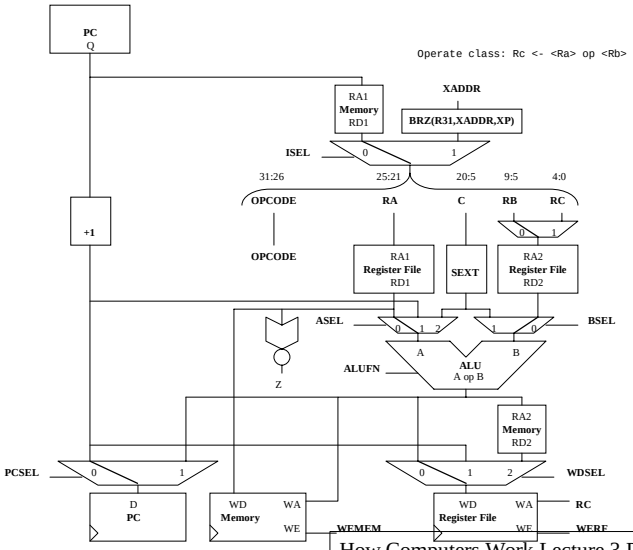
- **SHL, SHR, SAR**
  (shift left, right; shift arith right)

COMPARES

- **CMPEQ, CMPLT, CMPLE**

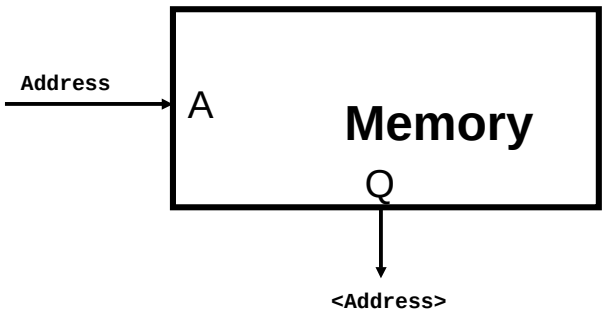
How Computers Work Lecture 3 Page 5

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 6

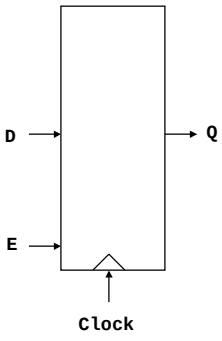
Combinational Read Port on Memory



How Computers Work Lecture 3 Page 7

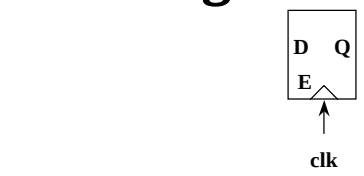
Data Register

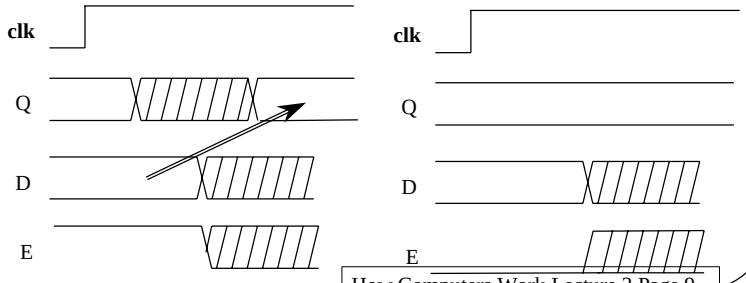
- Works like a camera
  - D = image
  - Q = picture
  - E = On/Off Switch
  - clock = shutter release button



How Computers Work Lecture 3 Page 8

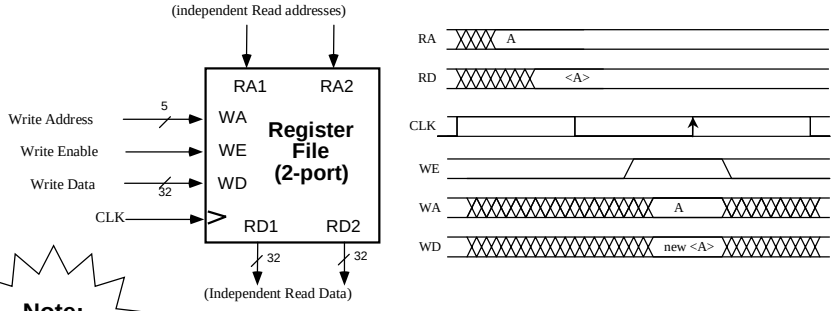
D Register w/ Enable





How Computers Work Lecture 3 Page 9

2-Port Register File



Note: <R31> Always ZERO!

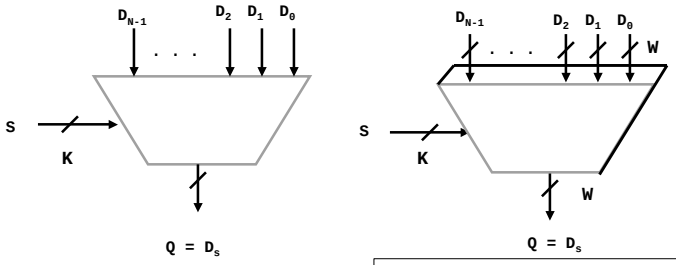
What if (say) WA=RA1???

RD1 reads value from last cycle!

How Computers Work Lecture 3 Page 10

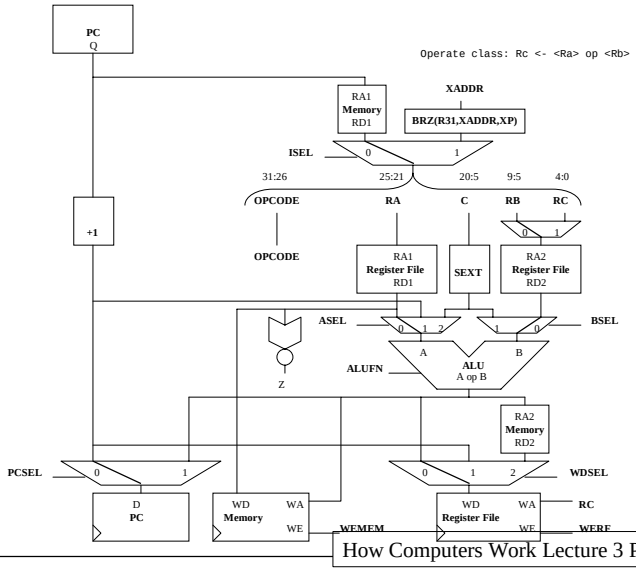
Selector (a.k.a. Multiplexor / MUX)

- Output Q is selected to be 1 of N inputs
- N is a power of 2
- K select inputs, K = log2(n)
- May be ganged to select one W-bit word out of N multi-bit words



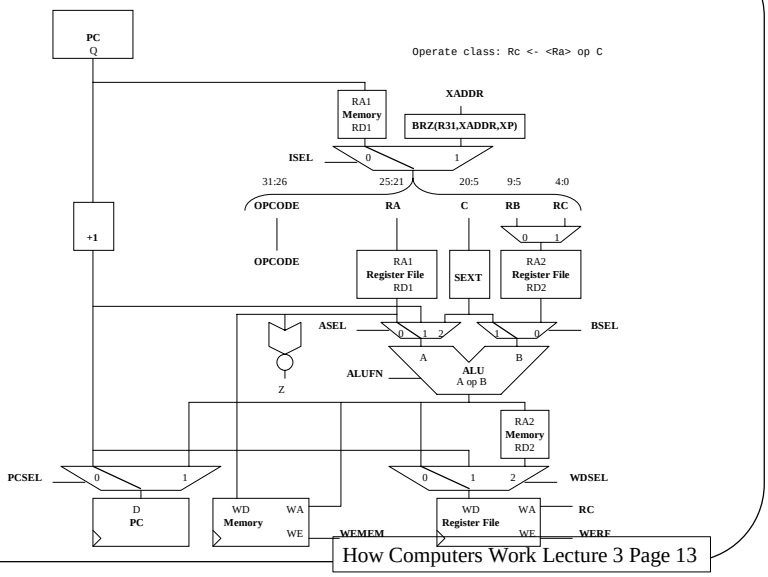
How Computers Work Lecture 3 Page 11

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 12

A Descending Data Flow View of the Beta



Review: β Branches

Conditional: **rc = <PC>+1; then**

BRNZ(ra, label, rc) **if <ra> nonzero then**
PC <- <PC> + displacement

BRZ(ra, label, rc) **if <ra> zero then**
PC <- <PC> + displacement

Unconditional: **rc = <PC>+1; then**

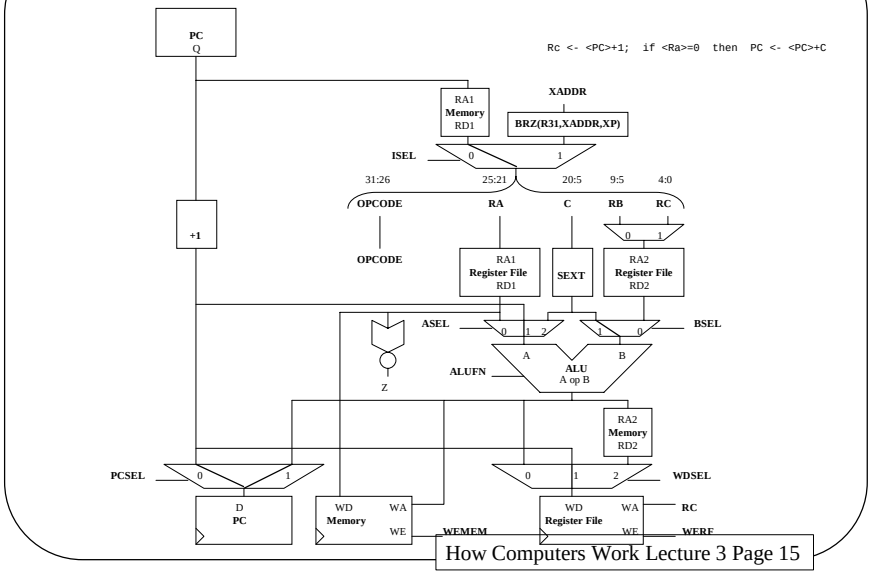
BRZ(r31, label, rc) **PC <- <PC> + displacement**

Indirect: **rc = <PC>+1; then**

JMP(ra, rc) **PC <- <ra>**

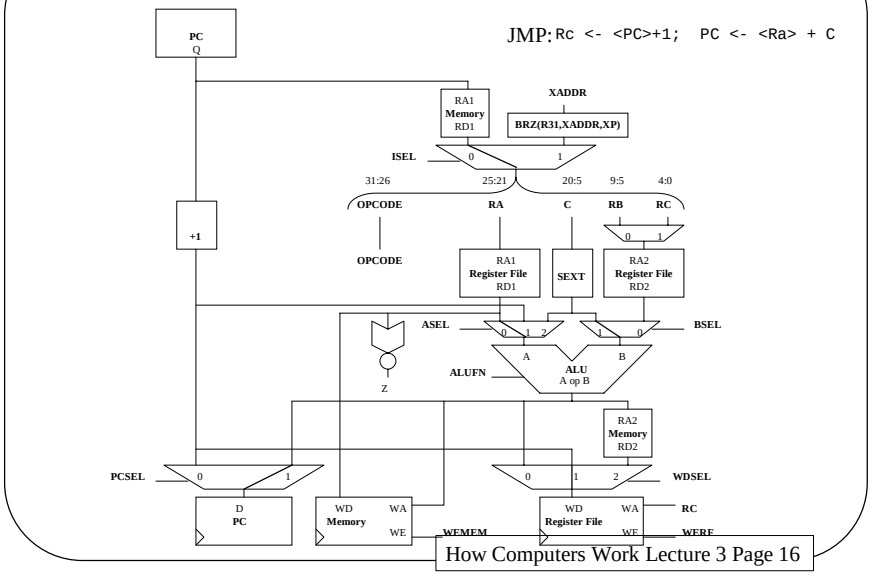
Note: "displacement" is coded as a CONSTANT in a field of the instruction!

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 15

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 16

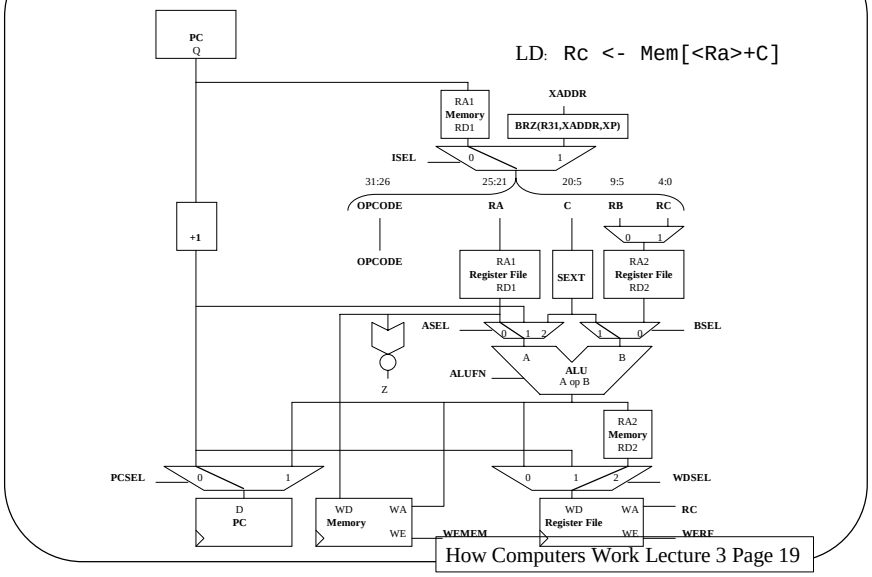
Review: β Loads & Stores

	LD(ra, C, rc)	**rc ← < Mem[<ra> + sext(C)] >**
`Old`	**ST(rc, C, ra)**	**Mem[<ra> + sext(C)] ← <rc>**
`New`	**ST(ra, C, rc)**	**Mem[<rc> + sext(C)] ← <ra>**

Straightening Out Store

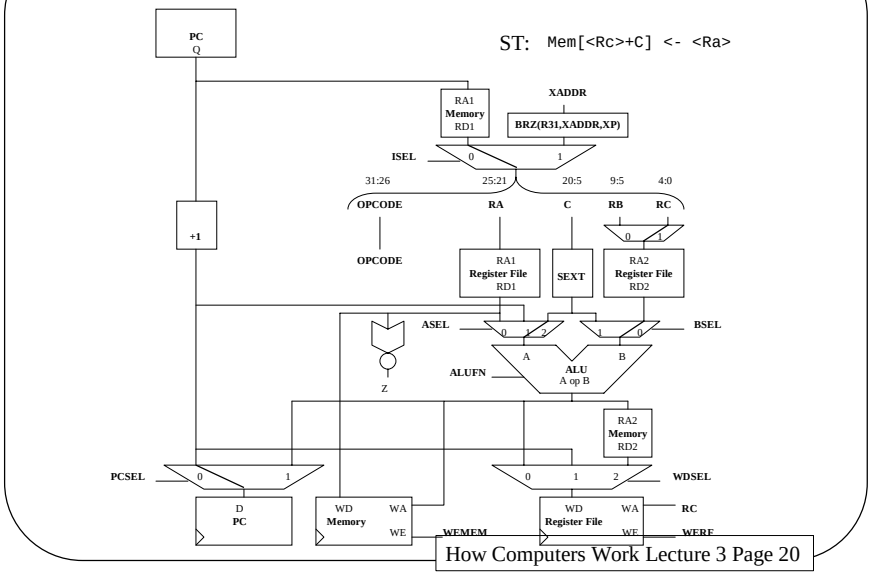
- Old Format: ST(Rc, C, Ra)
  - Mem[<Ra> + C] <- <Rc>
  - ST(R1, 2, R3) means Mem[<R3> + 2] <- <R1>
- New Format: ST(Ra, C, Rc)
  - Mem[<Rc> + C] <- <Ra>
  - ST(R1, 2, R3) means Mem[<R3> + 2] <- <R1>
- Both versions of Store work “from left to right” **in assembly language.**
- Difference is only in the **binary encoding** of the instruction, and the **hardware implementation’s decoding** of the binary encoding.

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 19

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 20

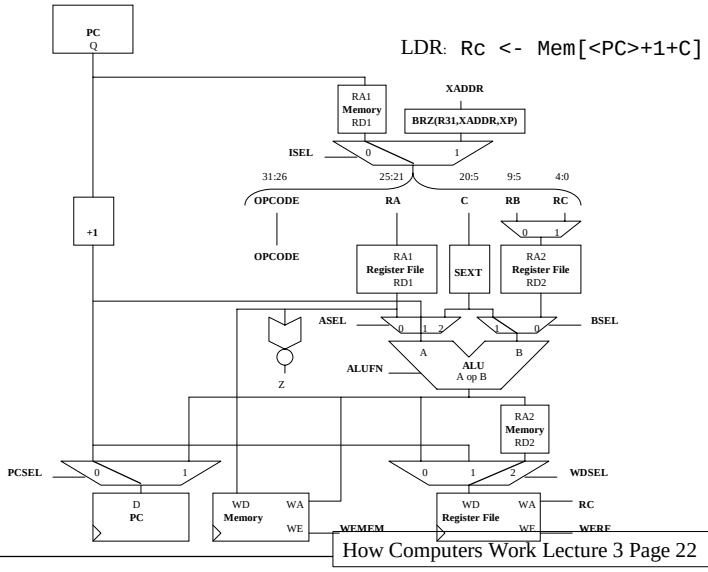
LDR
Load Relative

- Used for loading large (32 bit) constants with data from the instruction stream.
- Depends on the fact data and instruction memory are ports of one main memory.
- Use: LDR (label, Rc)
- RTL Description: Rc <- <Mem[<NextPC> + Offset]>
- Note that Ra is ignored, Offset is calculated from label

```
            …
            LDR (label, R1)
            BR (label + 1)
label:      123456789
            …
```

How Computers Work Lecture 3 Page 21

A Descending Data Flow View of the Beta



How Computers Work Lecture 3 Page 22

Control Logic Truth Table

We can specify it via a table of the form ...

Control Logic Inputs:

OPCODE	OP	OPC	LD	ST	BRZ		BRNZ		JMP	LDR	(Illegal)		
Z					0	1	0	1					
Control Logic Outputs:													
PCSEL													
RA2SEL													
ASEL													
BSEL													
WDSEL													
ALUFN													
Wr													
WERF													
WASEL													

YOU should be able to fill in this table!

Next Time - How to Add