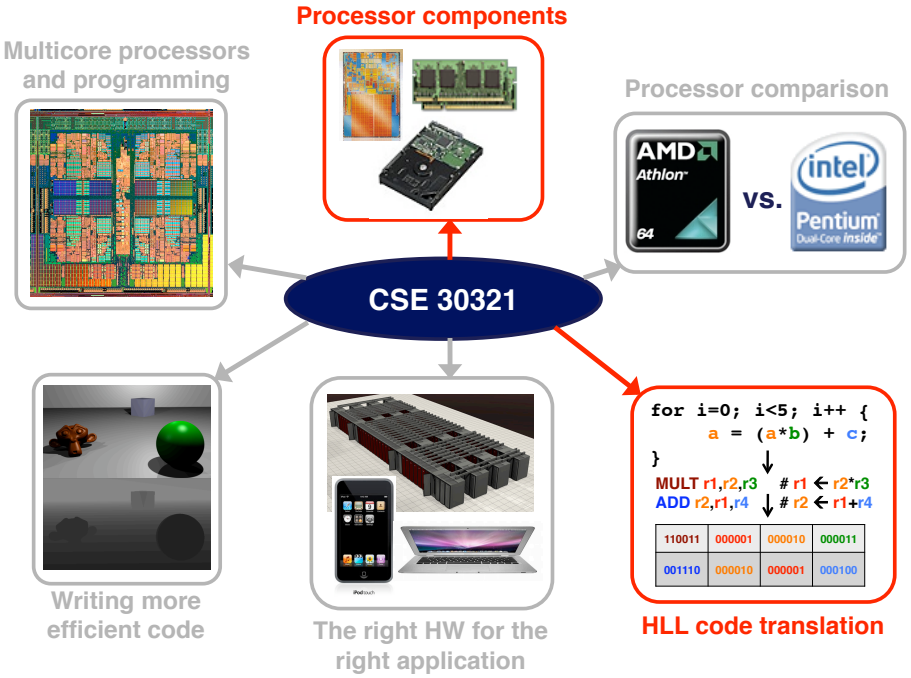
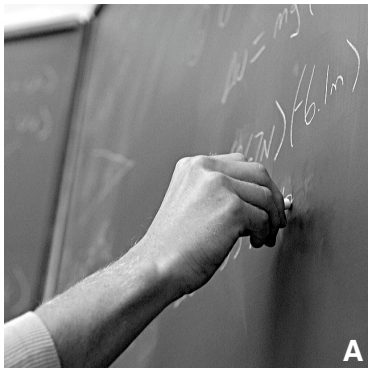
Lecture 02-03
Stored Programs

Some slides/images from Vahid text – hence this notice:



Board Discussion #1: Introduction to Stored Programs



Stored Program (summary)

A hypothetical translation:

```
for i=0; i<5; i++ {
      a = (a*b) + c;
}
```

```
MULT temp,a,b  # temp ← a*b
MULT r1,r2,r3  # r1 ← r2*r3
```

```
ADD a,temp,c   # a ← temp+c
ADD r2,r1,r4   # r2 ← r1+r4
```

Can define codes for MULT and ADD
Assume MULT = 110011 & ADD = 001110

stored program becomes

PC	110011	000001	000010	000011
PC+1	001110	000010	000001	000100

Things have not changed much...

- Stored program model has been around for a long time...

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

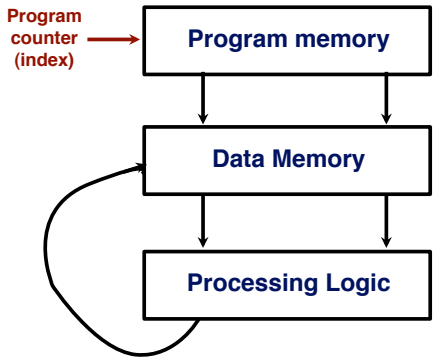
United States Army Ordnance Department

and the

University of Pennsylvania

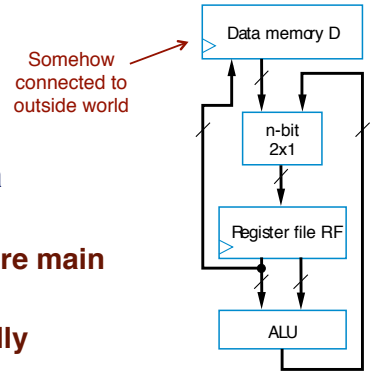
Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945



A simple "Von Neumann" architecture

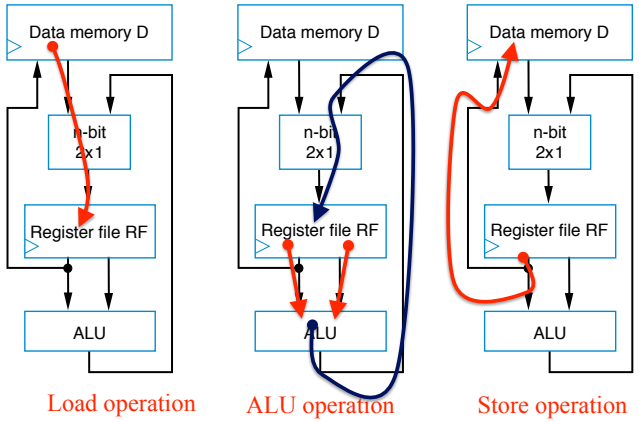
- Processing generally consists of:
  - Loading some data
  - Transforming that data
  - Storing that data
- Basic datapath: Useful circuit in a programmable processor
  - Can read/write data memory, where main data exists
  - Has register file to hold data locally
  - Has ALU to transform local data



Datapath

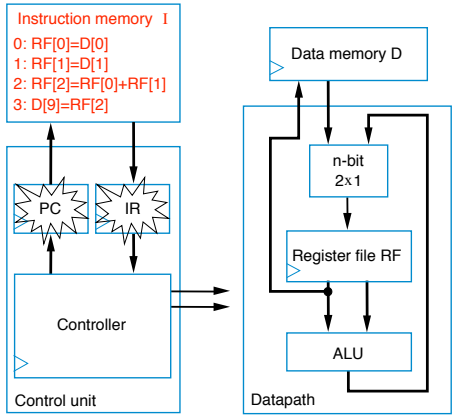
Basic Datapath Operations

- **Load operation:** Load data from data memory to RF
- **ALU operation:** Transforms data by passing one or two RF register values through ALU, performing operation (ADD, SUB, AND, OR, etc.), and writing back into RF.
- **Store operation:** Stores RF register value back into data memory
- Each operation can be done in one clock cycle



Basic Architecture – Control Unit

- D[9] = D[0] + D[1] – requires a sequence of four datapath operations:
  - 0: RF[0] = D[0]
  - 1: RF[1] = D[1]
  - 2: RF[2] = RF[0] + RF[1]
  - 3: D[9] = RF[2]
- Each operation is an *instruction*
  - Sequence of instructions – *program*
  - Looks cumbersome, but that's the world of programmable processors – Decomposing desired computations into processor-supported operations
  - Store program in *Instruction memory*
  - *Control unit* reads each instruction and executes it on the datapath
    - PC: Program counter – address of current instruction
    - IR: Instruc… instructio…

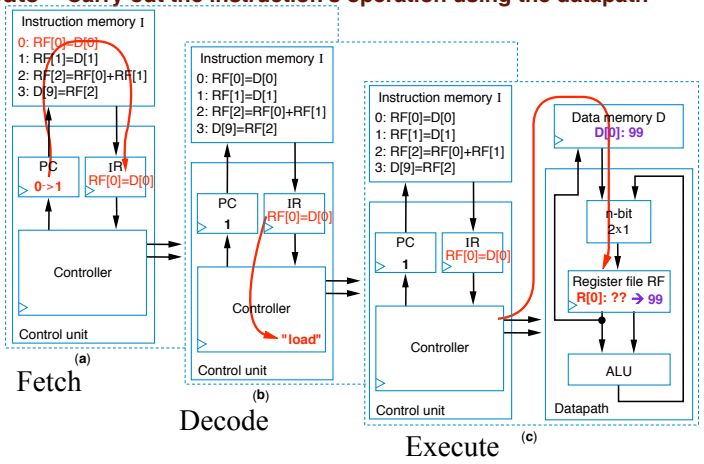


Digression:
HW vs. SW based approaches

Foreshadowing:
What if we want ALU to add, subtract?
How do we tell it what to do?

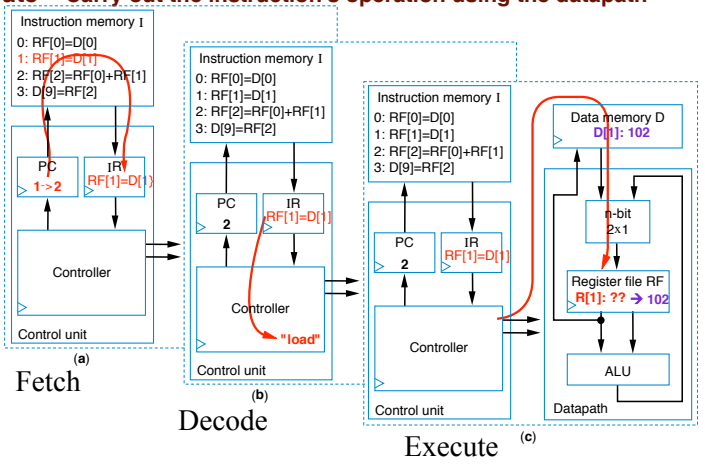
Basic Architecture – Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



Basic Architecture – Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



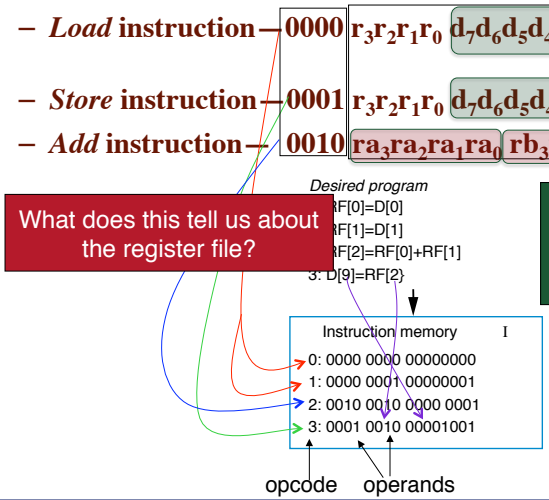
3-Instruction Programmable Processor

- Instruction Set – List of allowable inst
  representation in memory, e.g.,
  - *Load* instruction — 0000 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
  - *Store* instruction — 0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
  - *Add* instruction — 0010 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$

What does this tell you about data memory?

What does this tell us about the register file?

"Instruction" is an idea that helps abstract 1s, 0s, but still provides info. about HW



Instructions in 0s and 1s – *machine code*

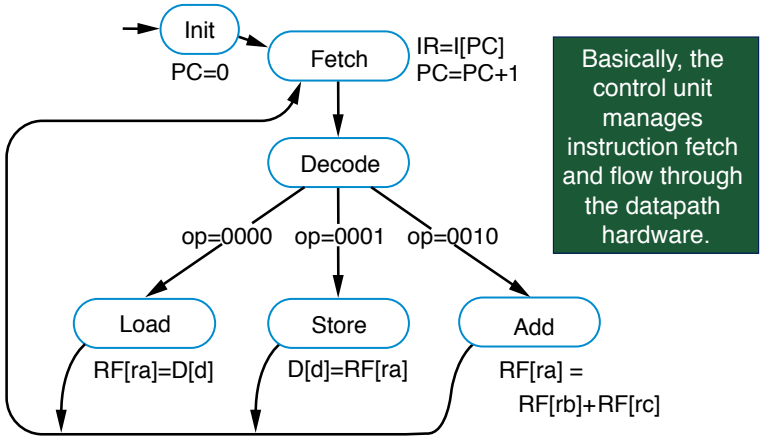
Assembly Code

- Machine code (0s and 1s) hard to work with
- Assembly code – Uses mnemonics
  - *Load* instruction — MOV Ra, d
    - specifies the operation *RF[a]=D[d]*. *a* must be 0,1, ..., or 15 — so *R0* means *RF[0]*, *R1* means *RF[1]*, etc. *d* must be 0, 1, ..., 255
  - • *Store* instruction — MOV d, Ra
    - specifies the operation *D[d]=RF[a]*
  - • *Add* instruction — ADD Ra, Rb, Rc
    - specifies the operation *RF[a]=RF[b]+RF[c]*

Desired program	machine code	assembly code
0: RF[0]=D[0]	0: 0000 0000 00000000	0: MOV R0, 0
1: RF[1]=D[1]	1: 0000 0001 00000001	1: MOV R1, 1
2: RF[2]=RF[0]+RF[1]	2: 0010 0010 0000 0001	2: ADD R2, R0, R1
3: D[9]=RF[2]	3: 0001 0010 00001001	3: MOV 9, R2

A control unit for the 3-Instruction Processor

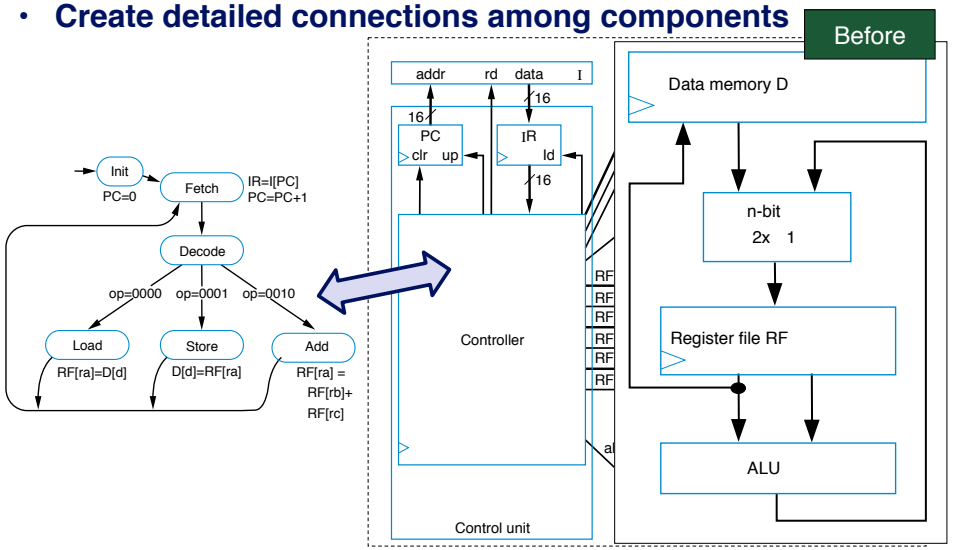
- To design the processor, we can begin with a high-level state machine description of the processor's behavior



Basically, the control unit manages instruction fetch and flow through the datapath hardware.

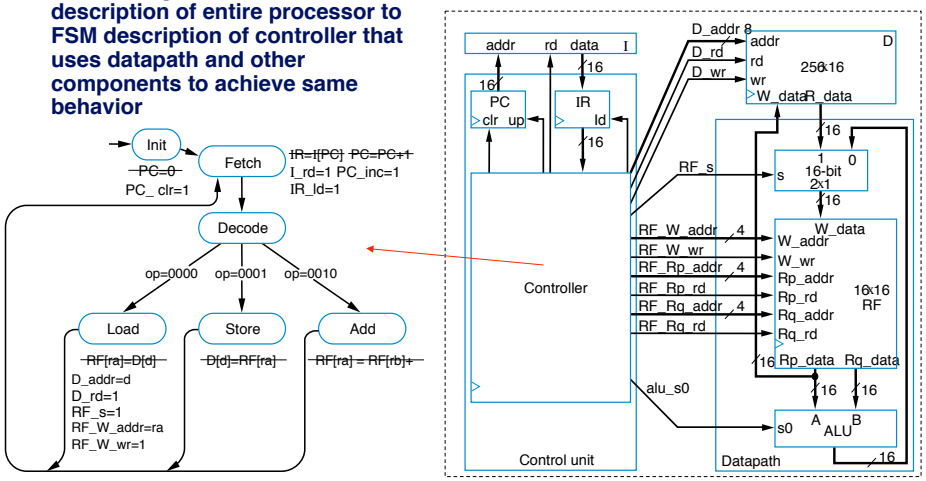
Control unit and datapath for 3-instruction processor

- Create detailed connections among components



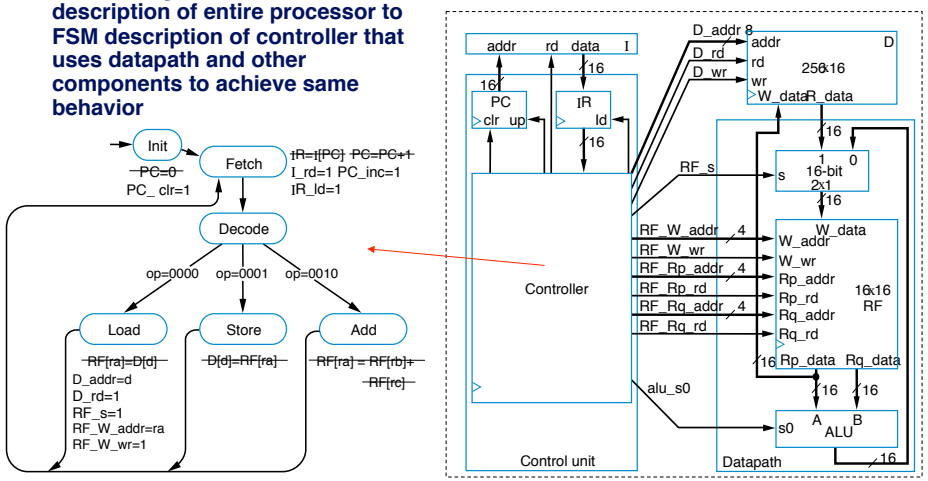
Control unit and datapath for 3-instruction processor

- Convert high-level state machine description of entire processor to FSM description of controller that uses datapath and other components to achieve same behavior



Control unit and datapath for 3-instruction processor

- Convert high-level state machine description of entire processor to FSM description of controller that uses datapath and other components to achieve same behavior



Be sure you understand the timing!

- Will the correct instruction be fetched if PC is incremented during the fetch cycle?
- While executing "MOV R1, 3", what is the content of PC and IR at the end of the 1st cycle, 2nd cycle, 3rd cycle, etc.? (assume we're at start of program)
- What if it takes more than 1 cycle for memory read?

Be sure you understand the timing!

Q1: D[8] = D[8] + RF[1] + RF[4]

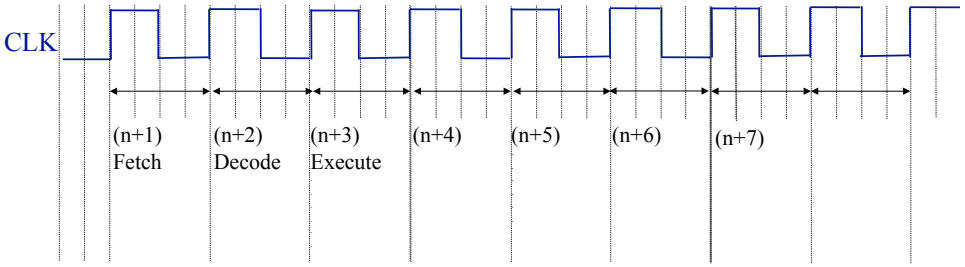
…

I[15]: Add R2, R1, R4 RF[1] = 4

I[16]: MOV R3, 8 RF[4] = 5

I[17]: Add R2, R2, R3 D[8] = 7

…



A 6-Instruction programmable processor

- **Let's add three more instructions:**
  - ***Load-constant* instruction—0011 $r_3r_2r_1r_0$ $c_7c_6c_5c_4c_3c_2c_1c_0$**
    - **MOV Ra, #c—specifies the operation *RF[a]=c***
  - ***Subtract* instruction—0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**
    - **SUB Ra, Rb, Rc—specifies the operation *RF[a]=RF[b] – RF[c]***
  - ***Jump-if-zero* instruction—0101 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$**
    - **JMPZ Ra, offset—specifies the operation *PC = PC + offset* if *RF[a]* is 0**

Adding instructions enables more sophisticated operations

TABLE 8.1 Six-instruction instruction set..

Instruction	Meaning
MOV Ra, d	RF[a] = D[d]
MOV d, Ra	D[d] = RF[a]
ADD Ra, Rb, Rc	RF[a] = RF[b]+RF[c]
MOV Ra, #C	RF[a] = C
SUB Ra, Rb, Rc	RF[a] = RF[b]-RF[c]
JMPZ Ra, offset	PC=PC+offset if RF[a]=0

TABLE 8.2 Instruction opcodes.

Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

Example Program

Compare the contents of D[4] and D[5].
If equal, D[3] =1, otherwise set D[3]=0.

```
     MOV   R0, #1        // RF[0] = 1
     MOV   R1, 4         // RF[1] = D[4]
     MOV   R2, 5         // RF[2] = D[5]
     SUB   R3, R1, R2    // RF[3] = RF[1]-RF[2]
     JMPZ  R3, B1        // if RF[3] = 0, jump to B1
     SUB   R0, R0, R0    // RF[0] = 0
B1:  MOV  3, R0          // D[3] = RF[0]
```

TABLE 8.2 Instruction opcodes.

Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

Program for the 6-Instruction Processor

- **Example program – Count number of non-zero words in D[4] and D[5]**
  - **Result will be either 0, 1, or 2**
  - **Put result in D[9]**

TABLE 8.2 Instruction opcodes.

Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

Modifications to 3-instruction processor

- ***Load-constant* instruction**

  0011 $r_3r_2r_1r_0c_7c_6c_5c_4c_3c_2c_1c_0$

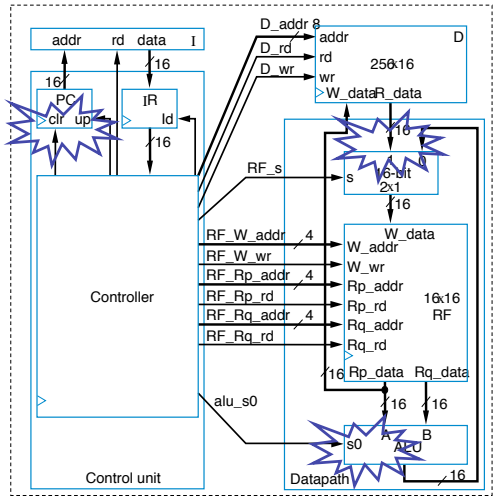
- ***Subtract* instruction**

  0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$

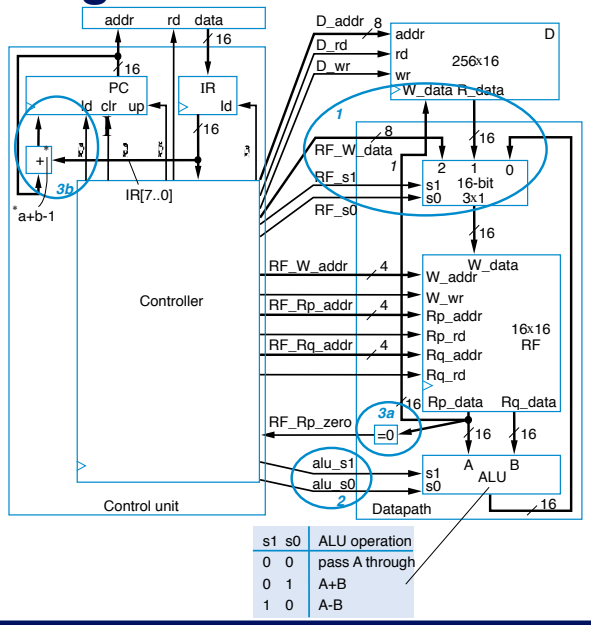
- ***Jump-if-zero* instruction**

  0101 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$

Adding instructions can also mean adding hardware



Extending the control unit and datapath



s1	s0	ALU operation
0	0	pass A through
0	1	A+B
1	0	A-B

Controller FSM for 6-instruction processor

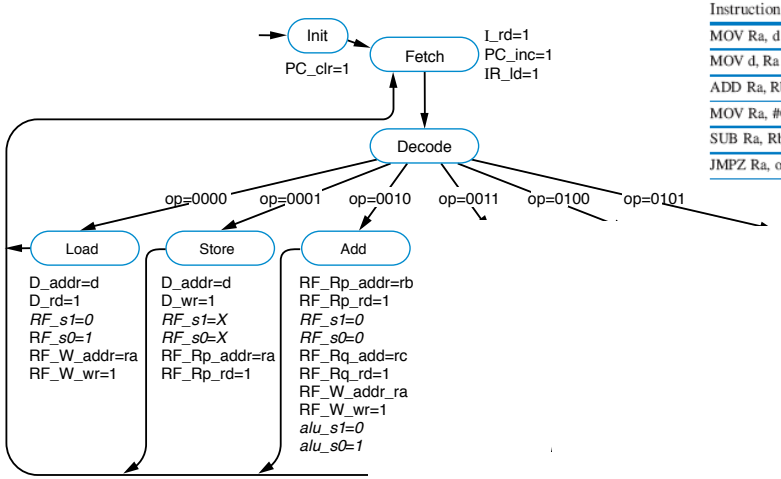


TABLE 8.2 Instruction opcodes.

Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

ARMs

Board Discussion #5: Wrap up, final examples