Lectures 02, 03
Introduction to Stored Programs
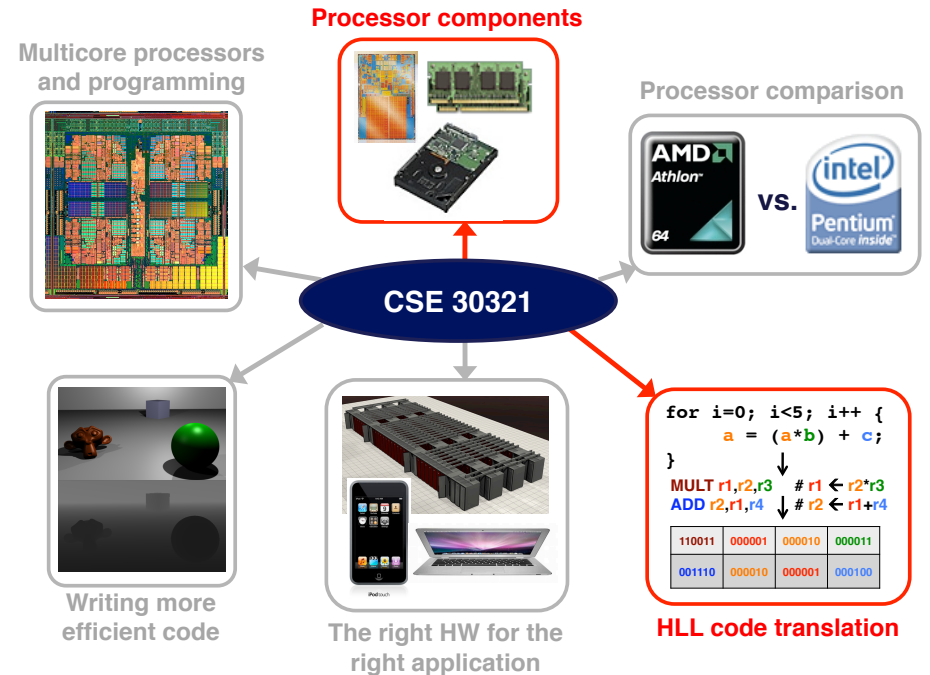
Suggested reading:
HP Chapters 1.1-1.3

Processor components

Multicore processors and programming

Processor comparison

CSE 30321

```
for i=0; i<5; i++ {
    a = (a*b) + c;
}
MULT r1,r2,r3   # r1 ← r2*r3
ADD r2,r1,r4    # r2 ← r1+r4
```

| 110011 | 000001 | 000010 | 000011 |
| 001110 | 000010 | 000001 | 000100 |

Writing more efficient code

The right HW for the right application

HLL code translation

Fundamental lesson(s)

- **How code you write (compiled C for example) is ultimately run on HW.**

Why it's important…

- **You'll learn what your microprocessor actually does when you compile and execute code written in a HLL.**

- **Equally important, at the heart of this discussion is the "stored program model".**
  - **This is a fundamental idea that we'll discuss over the entire semester … and many things build off of it.**

Board Discussion #1:
# Introduction to stored programs

A

Board discussion summary:

- **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
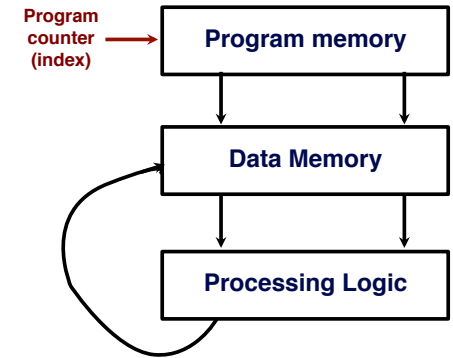
Board discussion summary:

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

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

stored program becomes {

A simple "Von Neumann" architecture

- **"Von Neumann architecture" synonymous with "programmable processor"**

- **Processing generally consists of:**
  - **Loading some data**
  - **Transforming that data**
  - **Storing that data**

- **Datapath:** core of a programmable processor
  - **Can read/write data memory**
  - **Has register file to hold subsets of memory**
    - **(in a local, fast memory)**
  - **Has ALU to transform local data**

Connected to other, peripheral HW

Data memory D

n-bit 2x1

Register file RF

ALU
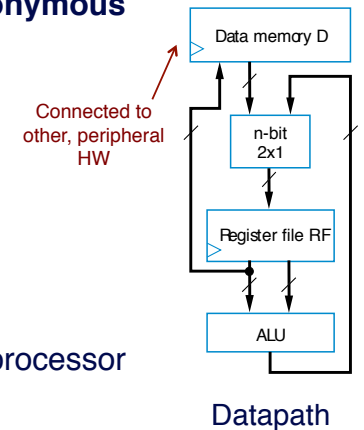
Datapath

Basic datapath operations

- **Load:** load data from data memory to RF
- **ALU operation:** transforms data by passing one or two RF values through ALU (for ADD, SUB, AND, OR, etc.); data written back to RF
- **Store operation:** stores RF register value back into data memory
- **Each operation can be done in one clock cycle**

Load operation   ALU operation   Store operation

B

9
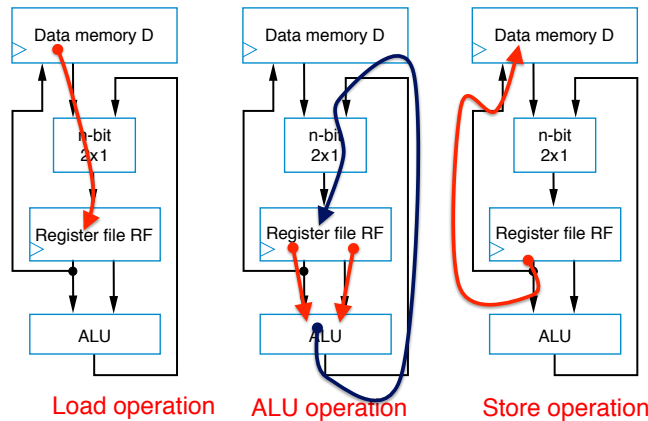
The datapath control unit

- 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**
  - 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:** Instruction register – current instruction
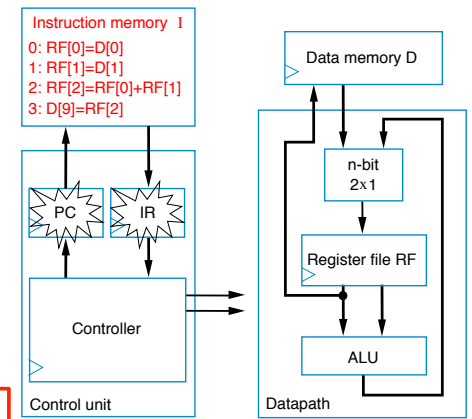
Foreshadowing:
What if we want ALU to add, subtract?
How do we tell it what to do?

10

The datapath control unit

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

Fetch   Decode   Execute
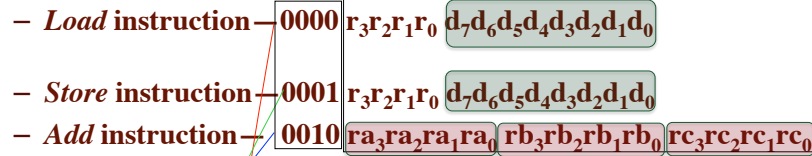
11

The datapath control unit

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

Fetch   Decode   Execute

12
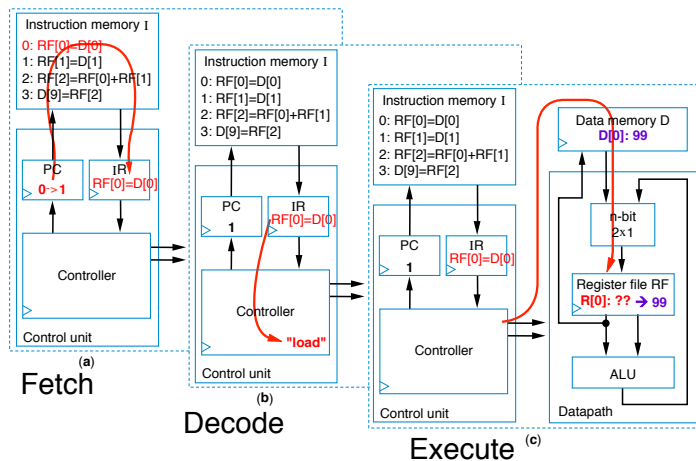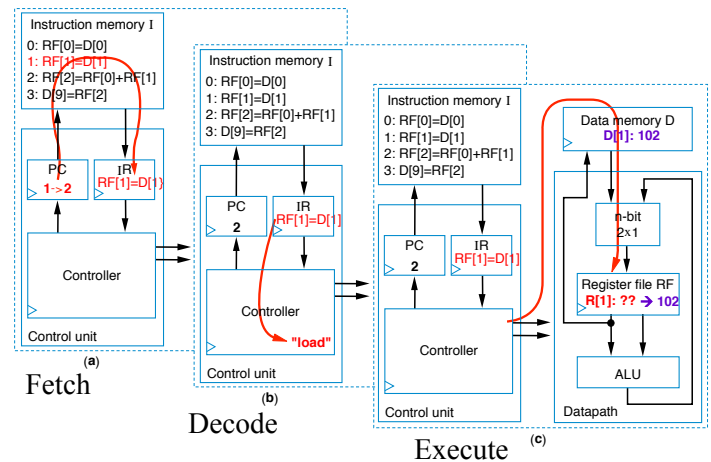
Datapath + control = 3-instruction programmable processor

- Instruction Set – List of allowable inst[ruction] representation in memory, e.g.,

What does this tell you about data memory?

- *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 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$
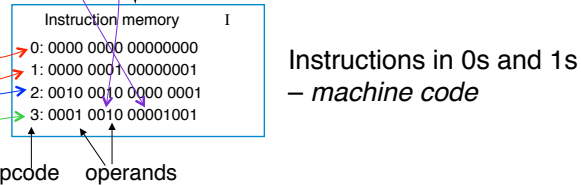
What does this tell us about the register file?

Desired program
RF[0]=D[0]
RF[1]=D[1]
RF[2]=RF[0]+RF[1]
3: D[9]=RF[2]

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

Instruction memory   I
0: 0000 0000 00000000
1: 0000 0001 00000001
2: 0010 0010 0000 0001
3: 0001 0010 00001001

opcode   operands

Instructions in 0s and 1s – *machine code*
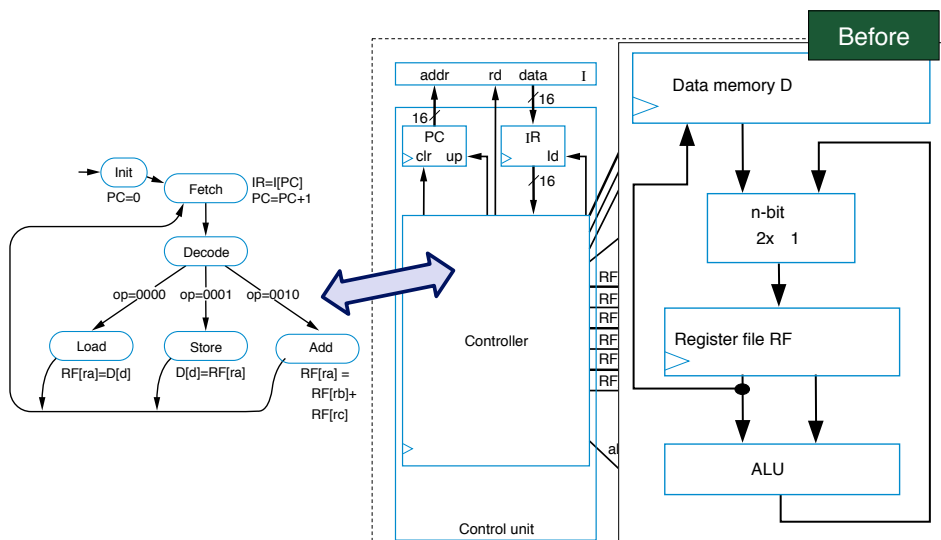
Toward a more detailed, realistic datapath…

- To design the processor, we can begin with a high-level state machine description of the processor's behavior
  - Control unit manages instruction fetch, flow through datapath HW

Init
PC=0

Fetch   IR=I[PC]
PC=PC+1

Decode

op=0000   op=0001   op=0010

Load   Store   Add

RF[ra]=D[d]   D[d]=RF[ra]   RF[ra] = RF[rb]+RF[rc]
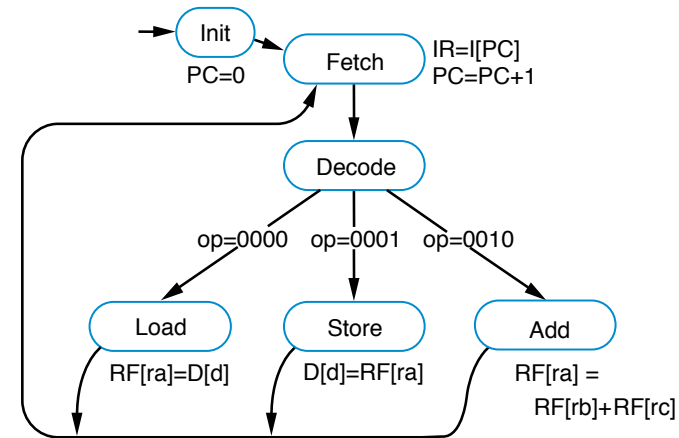
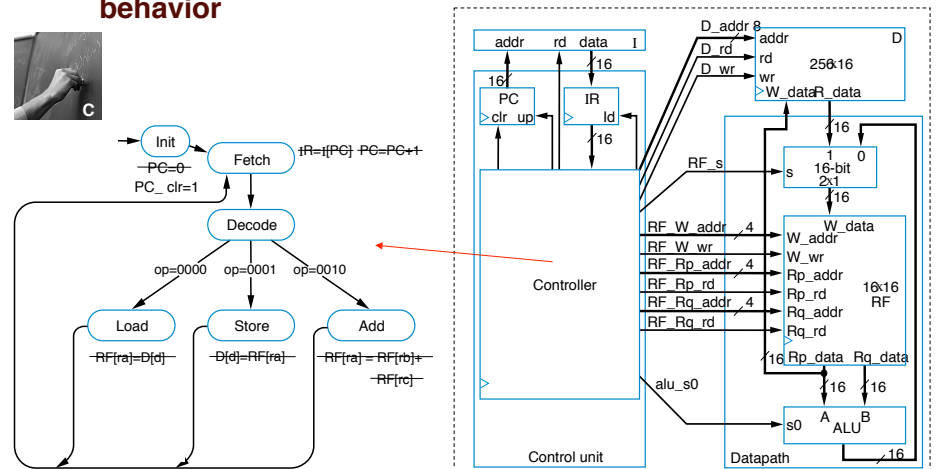## Toward a more detailed, realistic datapath…

- Now, create detailed connections among components

Before

Toward a more detailed, realistic datapath…

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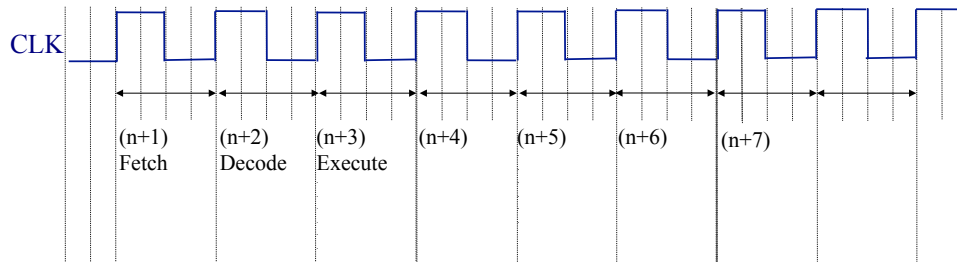 - Use datapath and other components to achieve same behavior

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

    ...

CLK

(n+1) Fetch   (n+2) Decode   (n+3) Execute   (n+4)   (n+5)   (n+6)   (n+7)

17

Assembly code (for 3-instruction processor)

- **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* is # between 0 and 15
      - R0 means RF[0], R1 means RF[1], etc.
      - d is # between 0 and 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
0: RF[0]=D[0]
1: RF[1]=D[1]
2: RF[2]=RF[0]+RF[1]
3: D[9]=RF[2]

0: 0000 0000 00000000   0: MOV R0, 0
1: 0000 0001 00000001   1: MOV R1, 1
2: 0010 0010 0000 0001   2: ADD R2, R0, R1
3: 0001 0010 00001001   3: MOV 9, R2

       machine code   assembly code

18

Important: 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?

D

19

A 6-Instruction programmable processor

- **Let's add three more (useful) 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**

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

20

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]
```

21

Program for the 6-Instruction processor

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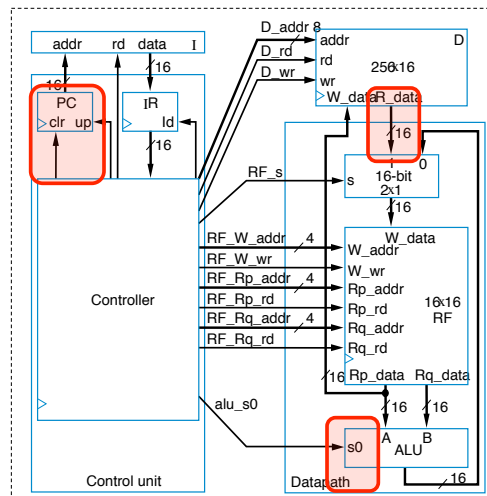 - **Result will be either 0, 1, or 2**
  - **Put result in D[9]**

TABLE 8.2 Instruction opcodes.

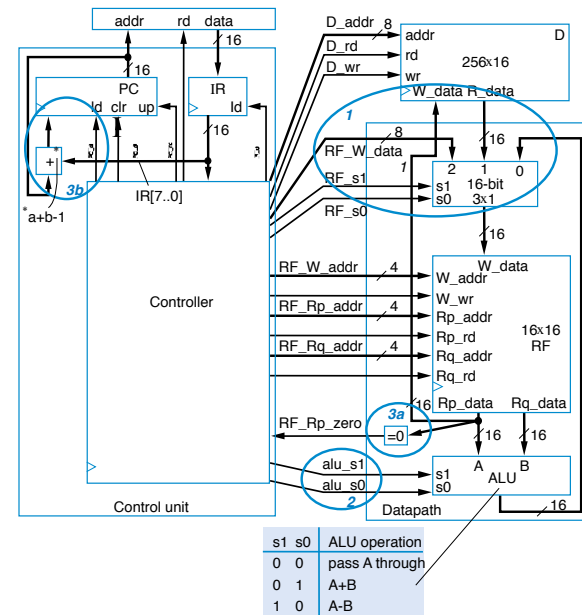| Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

E

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
  $0101\ ra_3ra_2ra_1ra_0$
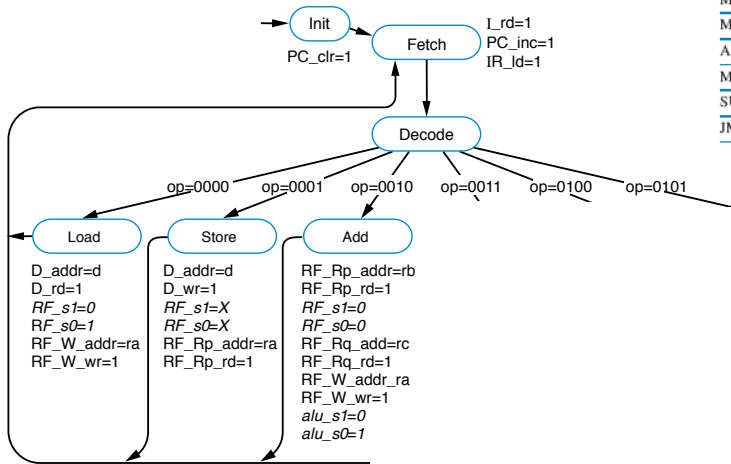 $o_7o_6o_5o_4o_3o_2o_1o_0$

Adding instructions can also mean adding hardware

23

Extending the control unit and datapath

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

F

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

FSM states:

Init → Fetch (PC_clr=1)

Fetch (I_rd=1, PC_inc=1, IR_ld=1) → Decode

Decode branches:
- op=0000 → Load
- op=0001 → Store
- op=0010 → Add
- op=0011
- op=0100
- op=0101

Load
D_addr=d
D_rd=1
RF_s1=0
RF_s0=1
RF_W_addr=ra
RF_W_wr=1

Store
D_addr=d
D_wr=1
RF_s1=X
RF_s0=X
RF_Rp_addr=ra
RF_Rp_rd=1

Add
RF_Rp_addr=rb
RF_Rp_rd=1
RF_s1=0
RF_s0=0
RF_Rq_add=rc
RF_Rq_rd=1
RF_W_addr_ra
RF_W_wr=1
alu_s1=0
alu_s0=1

25

HOW "REALISTIC" IS WHAT WE JUST DISCUSSED?

26

ARM7TDMI is real, commodity processor

http://www.st.com/stonline/products/families/evaluation_boards/industrial/factory_automation/motion_control/steval-ifn002v1/img/image_steval-ifn002v1.jpg
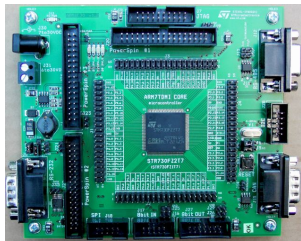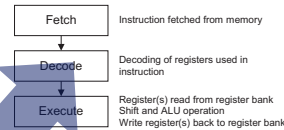
The ARM7TDMI core uses a pipeline to increase the speed of the flow of instructions to the processor. This enables several operations to take place simultaneously, and the processing and memory systems to operate continuously.

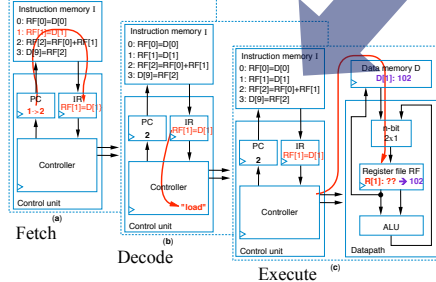A three-stage pipeline is used, so instructions are executed in three stages:
- Fetch
- Decode
- Execute.

The instruction pipeline is shown in Figure 1-1.

Source: ARM7TDMI Technical Reference Manual

Fetch	Instruction fetched from memory
Decode	Decoding of registers used in instruction
Execute	Register(s) read from register bank Shift and ALU operation Write register(s) back to register bank

Very similar to instruction execution stages just discussed

27

ARM7TDMI is real, commodity processor
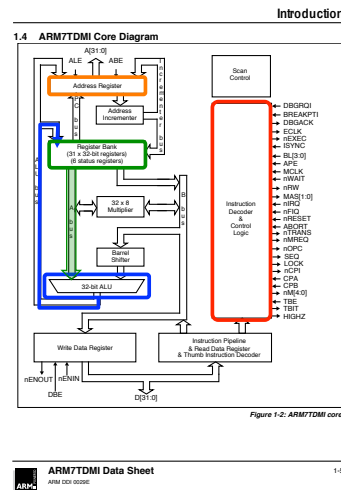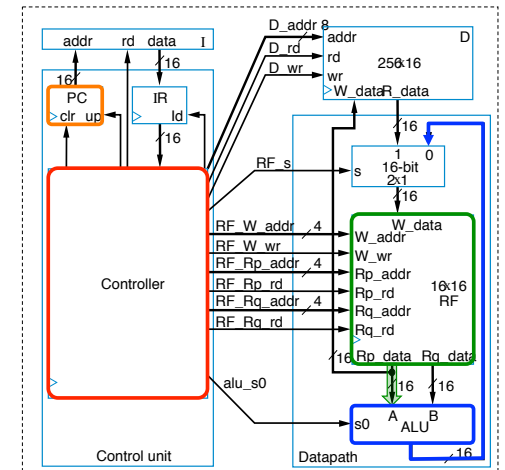
Very similar to instruction execution stages just discussed

28

Where is it used?

B&N Nook E-Reader

Microsoft Xbox 360 Wireless Steering Wheel

Nokia 500 Navigation

Fuji xerox DocuPrint C2090FS Colour Printer

Ugobe Pleo Dinosaur

ExaDigm XD2100SP Mobile Payment system

Triworks BEAUTY RF PLUS mod. BRF1

Over 10 billion units shipped.

http://www.arm.com/products/processors/classic/arm7/arm7tdmi.php

Board Discussion #5:
Wrap up, final examples

G