Digital Design

Chapter 8:

Programmable Processors

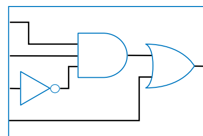
Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
http://www.ddvahid.com

Copyright © 2007 Frank Vahid

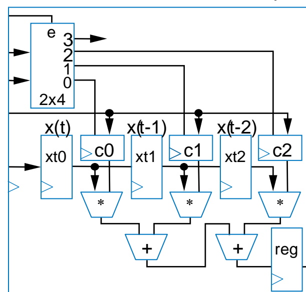
*Instructors of courses requiring Vahid's Digital Design textbook (published by John Wiley and Sons) have permission to modify and use these slides for customary course-related activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites.. PowerPoint source (or pdf with animations) may **not** be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see http://www.ddvahid.com for information.*

Introduction

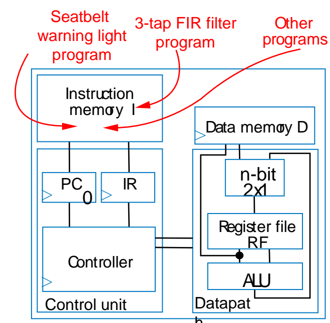
- Programmable (general-purpose) processor
  - Mass-produced, then programmed to implement different processing tasks
    - Well-known common programmable processors: Pentium, Sparc, PowerPC
    - Lesser-known but still common: ARM, MIPS, 8051, PIC
      - Low-cost embedded processors found in cell phones, blinking shoes, etc.
  - Instructive to design a very simple programmable processor
    - Real processors can be much more complex



Seatbelt warning light single-purpose processor



3-tap FIR filter single-purpose processor

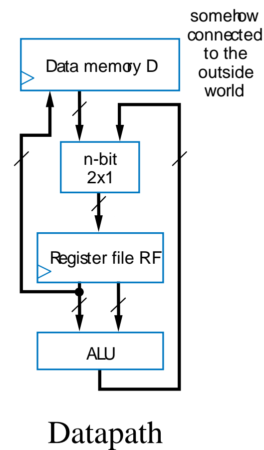


General-purpose processor

Note: Slides with animation are denoted with a small red "a" near the animated items

Basic 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

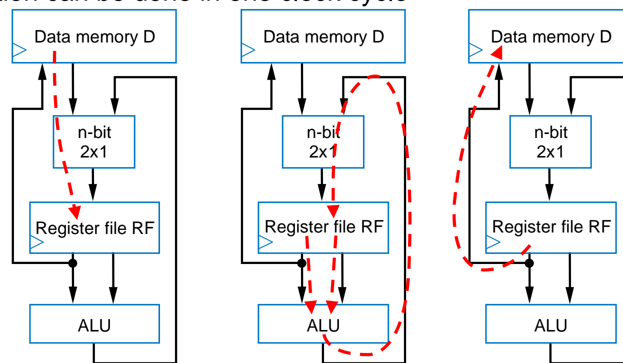


Datapath

Digital Design
Copyright © 2006
Frank Vahid

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

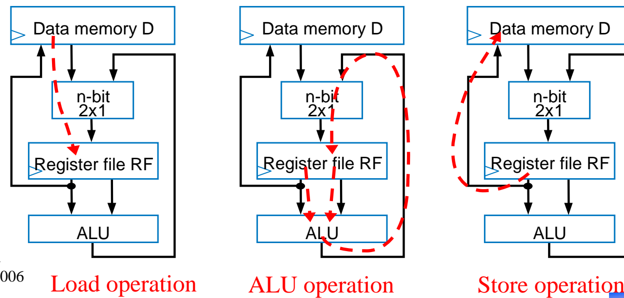


Load operation ALU operation Store operation

Digital Design
Copyright © 2006
Frank Vahid

Basic Datapath Operations

- Q: Which are valid *single-cycle operations* for given datapath?
  - Move D[1] to RF[1] (i.e., RF[1] = D[1])
    - A: YES – That's a load operation
  - Store RF[1] to D[9] and store RF[2] to D[10]
    - A: NO – Requires two separate store operations
  - Add D[0] plus D[1], store result in D[9]
    - A: NO – ALU operation (ADD) only works with RF. Requires two load operations (e.g., RF[0]=D[0]; RF[1]=D[1], an ALU operation (e.g., RF[2]=RF[0]+RF[1]), and a store operation (e.g., D[9]=RF[2])

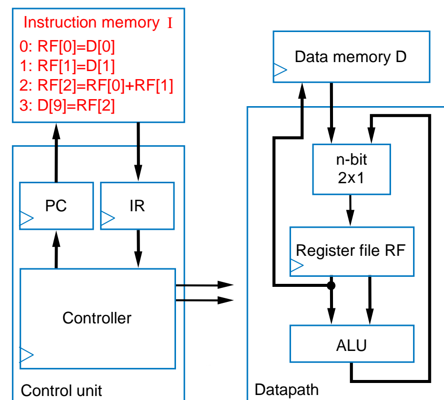


Load operation ALU operation Store operation

Digital Design
Copyright © 2006
Frank Vahid

Basic Architecture – Control Unit

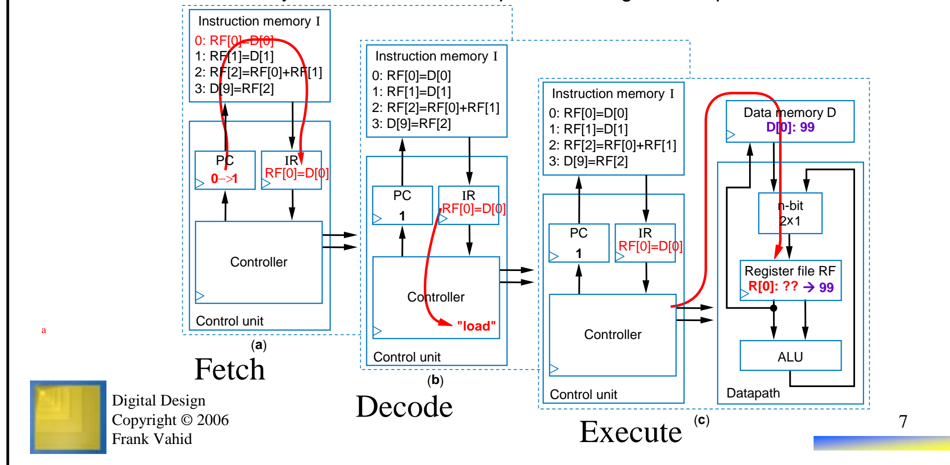
- D[9] = D[0] + D[1] – requires a sequence of four datapath operations:
  0: RF[0] = D[0]
  1: RF[1] = D[1]
  2: RF[2] = RF[0] + RF[1]
  3: D[9] = RF[2]
- Each operation is an *instruction*
  - Sequence of instructions – *program*
  - Looks cumbersome, but that's the world of programmable processors – 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: Instruction register – current instruction



Digital Design
Copyright © 2006
Frank Vahid

Basic Architecture – Control Unit

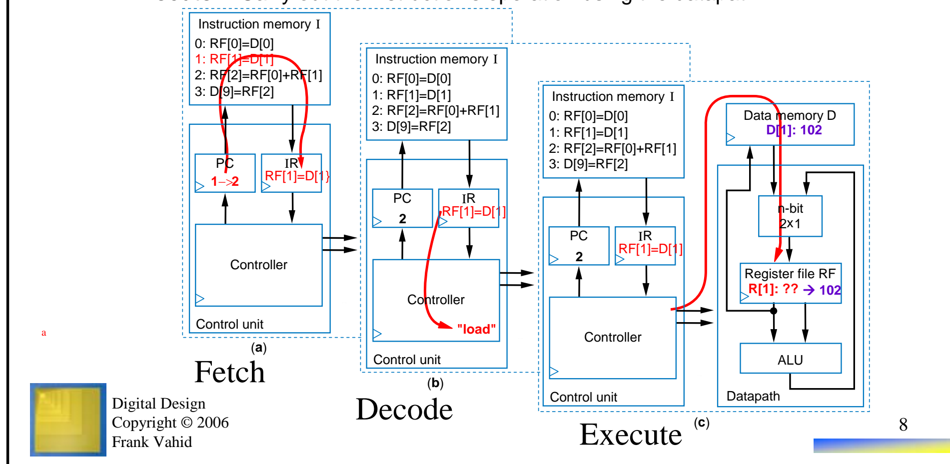
- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



Digital Design
Copyright © 2006
Frank Vahid

Basic Architecture – Control Unit

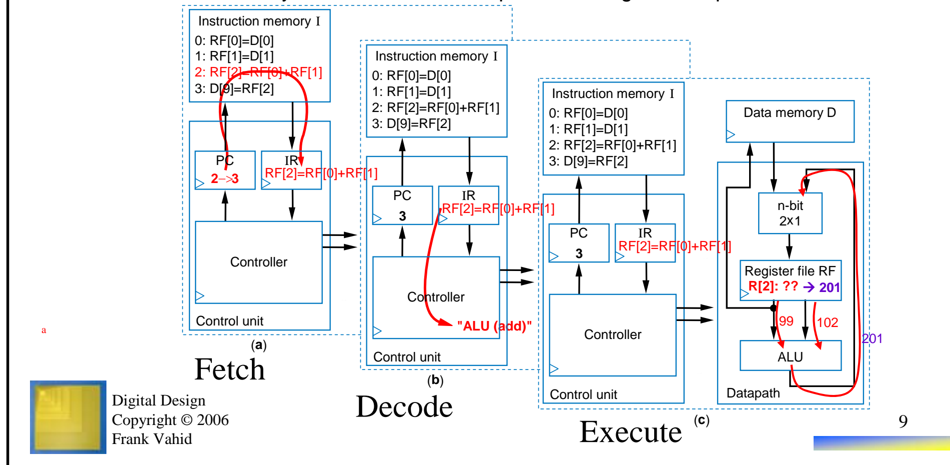
- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



Digital Design
Copyright © 2006
Frank Vahid

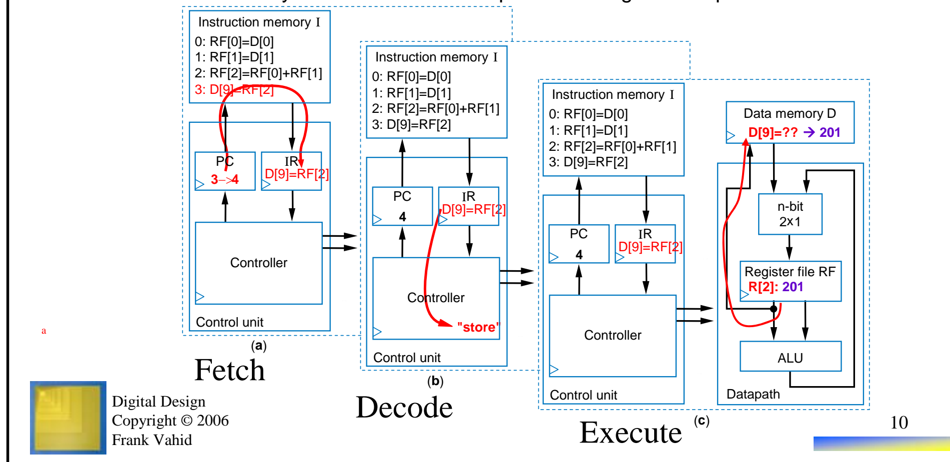
Basic Architecture – Control Unit

- 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

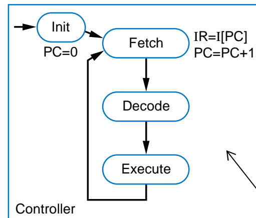
- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath

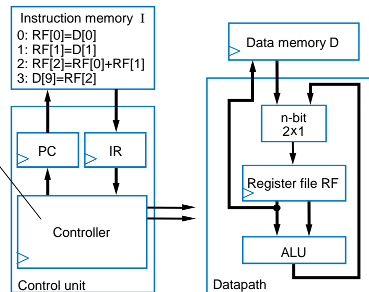


Basic Architecture – Control Unit

To summarize, the control unit processes each instruction in three stages:

1. first *fetching* the instruction by loading the current instruction into *IR* and incrementing the *PC* for the next fetch,
2. next *decoding* the instruction to determine its operation, and
3. finally *executing* the operation by setting the appropriate control lines for the datapath, if applicable. If the operation is a datapath operation, the operation may be one of three possible types:
   (a) *loading* a data memory location into a register file location,
   (b) transforming data using an *ALU* operation on register file locations and writing results back to a register file location, or
   (c) *storing* a register file location into a data memory location.





Creating a Sequence of Instructions

- Q: Create sequence of instructions to compute D[3] = D[0]+D[1]+D[2] on earlier-introduced processor
- A1: One possible sequence
  - First load data memory locations into register file
    - R[3] = D[0]
    - R[4] = D[1]
    - R[2] = D[2]

    (Note arbitrary register locations)
  - Next, perform the additions
    - R[1] = R[3] + R[4]
    - R[1] = R[1] + R[2]
  - Finally, store result
    - D[3] = R[1]
- A2: Alternative sequence
  - First load D[0] and D[1] and add them
    - R[1] = D[0]
    - R[2] = D[1]
    - R[1] = R[1] + R[2]
  - Next, load D[2] and add
    - R[2] = D[2]
    - R[1] = R[1] + R[2]
  - Finally, store result
    - D[3] = R[1]

Number of Cycles

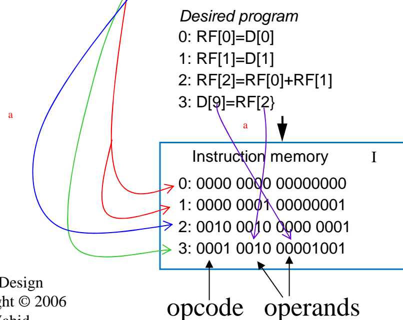
- Q: How many cycles are needed to execute six instructions using the earlier-described processor?
- A: Each instruction requires 3 cycles – 1 to fetch, 1 to decode, and 1 to execute
  - Thus, 6 instr * 3 cycles/instr = 18 cycles

Digital Design
Copyright © 2006 Frank Vahid

8.3

Three-Instruction Programmable Processor

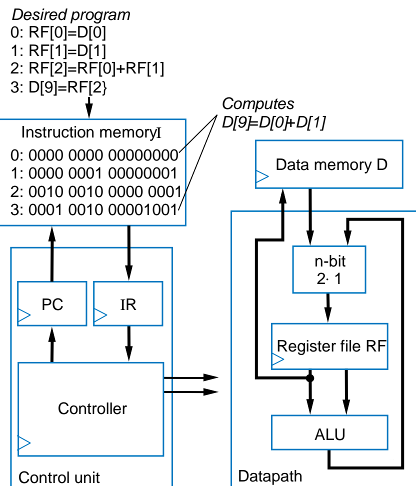
- Instruction Set – List of allowable instructions and their representation in memory, e.g.,
  - ***Load*** instruction—**0000** $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
  - ***Store*** instruction—**0001** $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
  - ***Add*** instruction—**0010** $ra_3ra_2ra_1ra_0$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



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

Digital Design
Copyright © 2006 Frank Vahid

Program for Three-Instruction Processor



Digital Design
Copyright © 2006
Frank Vahid

Program for Three-Instruction Processor

- Another example program in machine code
  - Compute D[5] = D[5] + D[6] + D[7]

```
0: 0000 0000 00000101  // RF[0] = D[5]
1: 0000 0001 00000110  // RF[1] = D[6]
2: 0000 0010 00000111  // RF[2] = D[7]
3: 0010 0000 0000 0001  // RF[0] = RF[0] + RF[1]
                        // which is D[5]+D[6]
4: 0010 0000 0000 0010  // RF[0] = RF[0] + RF[2]
                        // now D[5]+D[6]+D[7]
5: 0001 0000 00000101  // D[5] = RF[0]
```

–***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 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

Digital Design
Copyright © 2006
Frank Vahid

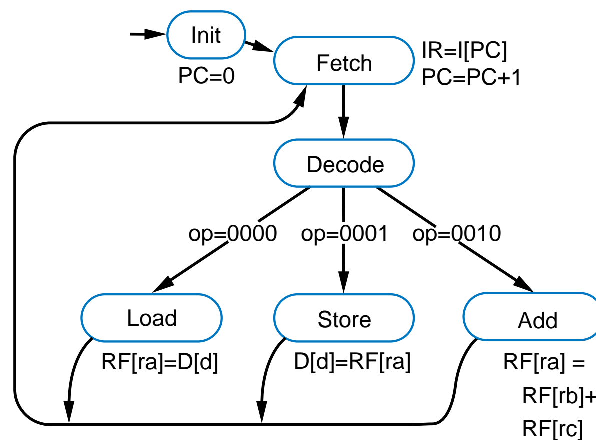
Assembly Code

- 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

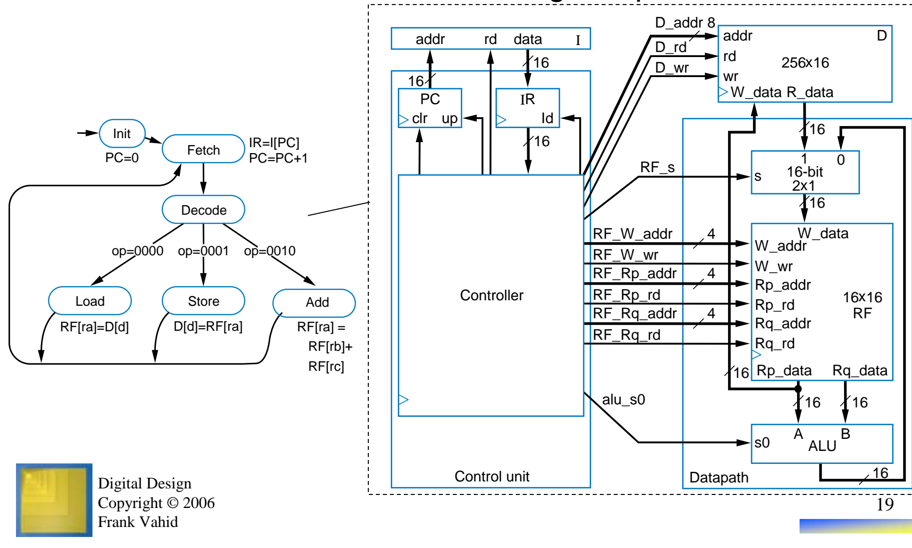
Control-Unit and Datapath for Three-Instruction Processor

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



Control-Unit and Datapath for Three-Instruction Processor

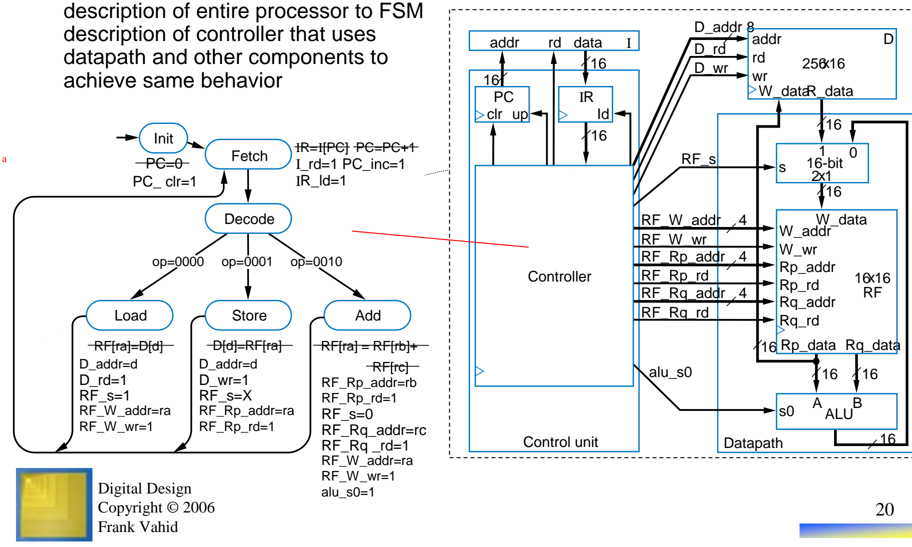
- Create detailed connections among components



Digital Design
Copyright © 2006
Frank Vahid

Control-Unit and Datapath for Three-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



Digital Design
Copyright © 2006
Frank Vahid

8.4

A Six-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

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

Digital Design
Copyright © 2006
Frank Vahid

Extending the Control-Unit and Datapath

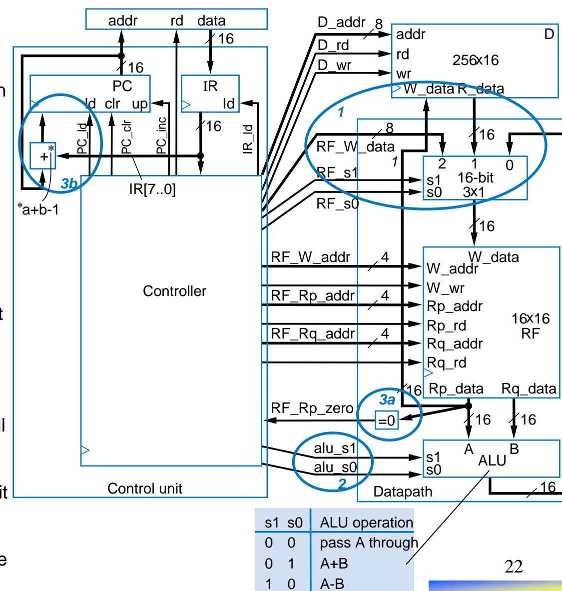
1: The *load constant* instruction requires that the register file be able to load data from *IR[7..0]*, in addition to data from data memory or the ALU output. Thus, we widen the register file's multiplexer from 2x1 to 3x1, add another mux control signal, and also create a new signal coming from the controller labeled *RF_W_data*, which will connect with *IR[7..0]*.

2: The subtract instruction requires that we use an ALU capable of subtraction, so we add another ALU control signal.

3: The jump-if-zero instruction requires that we be able to detect if a register is zero, and that we be able to add *IR[7..0]* to the *PC*.

3a: We insert a datapath component to detect if the register file's *Rp* read port is all zeros (that component would just be a NOR gate).

3b: We also upgrade the *PC* register so it can be loaded with *PC* plus *IR[7..0]*. The adder used for this also subtracts 1 from the sum, to compensate for the fact that the *Fetch* state already added 1 to the *PC*.



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

Digital Design
Copyright © 2006
Frank Vahid

Controller FSM for the Six-Instruction Processor

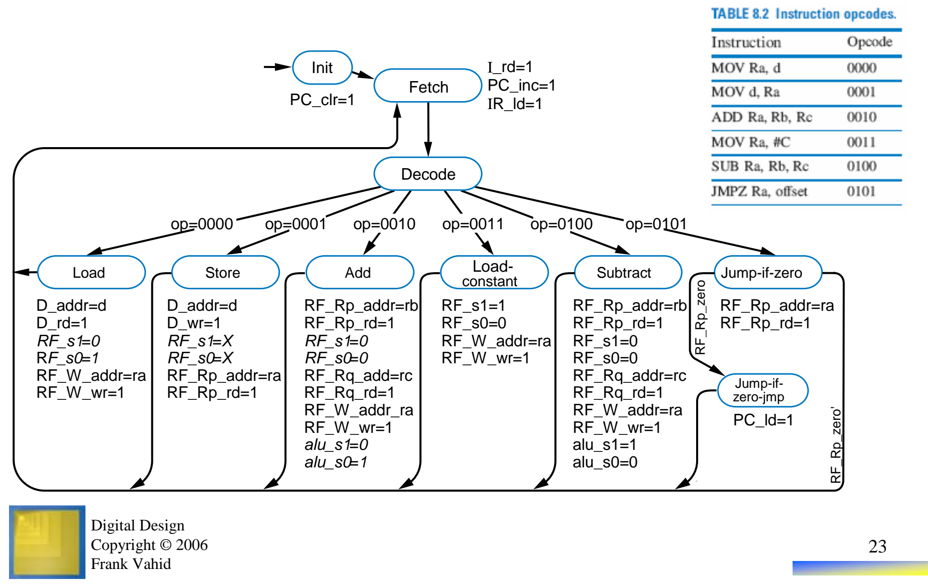


TABLE 8.2 Instruction opcodes.

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

Program for the Six-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]

```
      MOV R0, #0; // initialize result to 0                0011 0000 00000000
      MOV R1, #1; // constant 1 for incrementing result    0011 0001 00000001
      MOV R2, 4; // get data memory location 4             0000 0010 00000100
      JMPZ R2, lab1; // if zero, skip next instruction     0101 0010 00000010
      ADD R0, R0, R1; // not zero, so increment result     0010 0000 0000 0001
lab1:MOV R2, 5; // get data memory location 5              0000 0010 00000101
      JMPZ R2, lab2; // if zero, skip next instruction     0101 0010 00000010
      ADD R0, R0, R1; //not zero, so increment result      0010 0000 0000 0001
lab2:MOV 9, R0; // store result in data memory location 9  0001 0000 00001001
```

(a) (b)

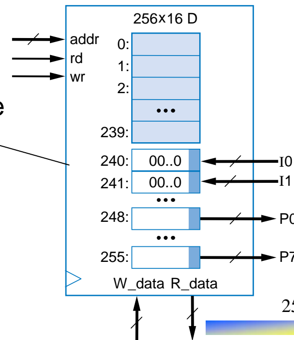
TABLE 8.2 Instruction opcodes.

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

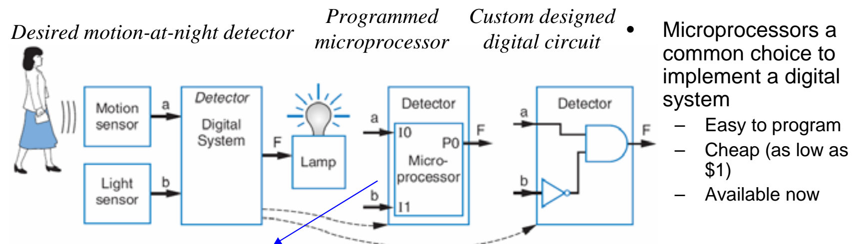
Digital Design
Copyright © 2006
Frank Vahid

Further Extensions to the Programmable Processor

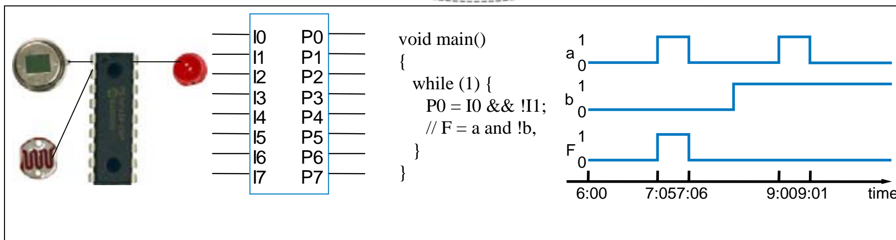
- Typical processor instruction set will contain dozens of data movement (e.g., loads, stores), ALU (e.g., add, sub), and flow-of-control (e.g., jump) instructions
  - Extending the control-unit/datapath follows similarly to previously-shown extensions
- Input/output extensions
  - Certain memory locations may actually be external pins
    - e.g, D[240] may represent 8-bit input I0, D[255] may represent 8-bit output P7



Program using I/O Extensions – Recall Chpt 1 C-Program Example



- Microprocessors a common choice to implement a digital system
  - Easy to program
  - Cheap (as low as $1)
  - Available now

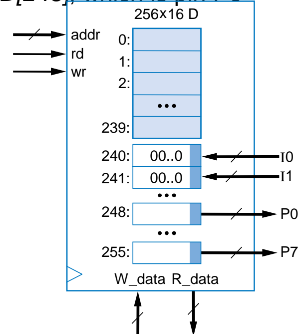


Program Using Input/Output Extensions

Underlying assembly code for C expression I0 && !I1.

0: MOV R0, 240 // move *D[240]*, which is the value at pin *I0*, into *R0*
1: MOV R1, 241 // move *D[241]*, which is that value at pin *I1*, into *R1*
2: NOT R1, R1 // compute !*I1*, assuming existence of a complement instruction
3: AND R0, R0, R1 // compute *I0* && !*I1*, assuming an AND instruction
4: MOV 248, R0 // move result to *D[248]*, which is pin *P0*

```
void main()
{
  while (1) {
    P0 = I0 && !I1;
    // F = a and !b,
  }
}
```



Digital Design
Copyright © 2006
Frank Vahid

Chapter Summary

- Programmable processors are widely used
  - Easy availability, short design time
- Basic architecture
  - Datapath with register file and ALU
  - Control unit with PC, IR, and controller
  - Memories for instructions and data
  - Control unit fetches, decodes, and executes
- Three-instruction processor with machine-level programs
  - Extended to six instructions
  - Real processors have dozens or hundreds of instructions
  - Extended to access external pins
  - Modern processors are far more sophisticated
- Instructive to see how one general circuit (programmable processor) can execute variety of behaviors just by programming 0s and 1s into an instruction memory

Digital Design
Copyright © 2006
Frank Vahid