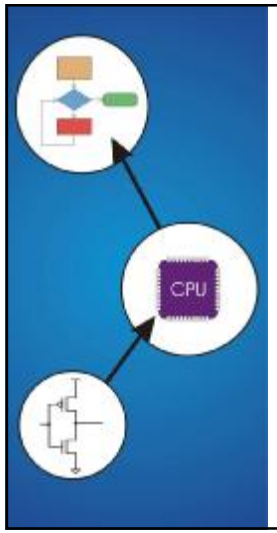


Chapter 5
The LC-2

Instruction Set Architecture

ISA = All of the *programmer-visible* components and operations of the computer

- **memory organization**
  - Ø **address space -- how may locations can be addressed?**
  - Ø **addressibility -- how many bits per location?**
- **register set**
  - Ø **how many? what size? how are they used?**
- **instruction set**
  - Ø **opcodes**
  - Ø **data types**
  - Ø **addressing modes**

ISA provides all information needed for someone that wants to write a program in machine language (or translate from a high-level language to machine language).

LC-2 Overview: Memory and Registers

Memory

- **address space: 2^{16} locations (16-bit addresses)**
- **addressibility: 16 bits**

Registers

- **temporary storage, accessed in a single machine cycle**
  - Ø **accessing memory generally takes longer than a single cycle**
- **eight general-purpose registers: R0 - R7**
  - Ø **each 16 bits wide**
  - Ø **how many bits to uniquely identify a register?**
- **other registers**
  - Ø **not directly addressible, but used by (and affected by) instructions**
  - Ø **PC (program counter), condition codes**

LC-2 Overview: Instruction Set

Opcodes

- **16 opcodes**
- ***Operate* instructions: ADD, AND, NOT**
- ***Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI**
- ***Control* instructions: BR, JSR, JSRR, RET, RTI, TRAP**
- **some opcodes set/clear *condition codes*, based on result:**
  - Ø **N = negative, Z = zero, P = positive (> 0)**

Data Types

- **16-bit 2's complement integer**

Addressing Modes

- **How is the location of an operand specified?**
- **non-memory addresses: *immediate*, *register***
- **memory addresses: *direct*, *indirect*, *base+offset***

Operate Instructions

Only three operations: ADD, AND, NOT

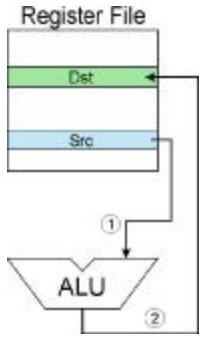
Source and destination operands are registers

- **These instructions *do not* reference memory.**
- **ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.**

Will show dataflow diagram with each instruction.

- **illustrates *when* and *where* data moves to accomplish the desired operation**

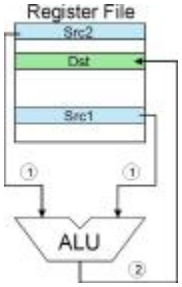
NOT (Register)



Note: Src and Dst could be the same register.

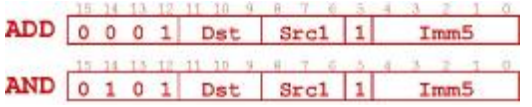
ADD/AND (Register)

this zero means "register mode"

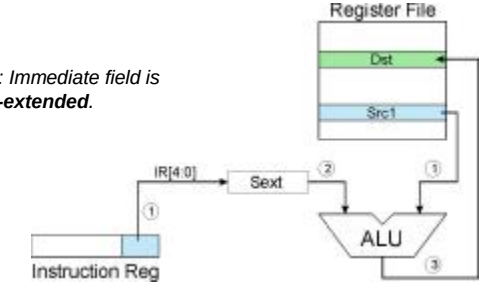


ADD/AND (Immediate)

this one means "immediate mode"



*Note: Immediate field is **sign-extended**.*



Using Operate Instructions

With only ADD, AND, NOT…

- **How do we subtract?**
- **How do we OR?**
- **How do we copy from one register to another?**
- **How do we initialize a register to zero?**

Data Movement Instructions

Load -- read data from memory to register

- **LD: direct mode**
- **LDR: base+offset mode**
- **LDI: indirect mode**

Store -- write data from register to memory

- **ST: direct mode**
- **STR: base+offset mode**
- **STI: indirect mode**

Load effective address -- compute address, save in register

- **LEA: immediate mode**
- ***does not access memory***

Direct Addressing Mode

Want to specify address directly in the instruction

- **But an address is 16 bits, and so is an instruction!**
- **After subtracting 4 bits for opcode and 3 bits for register, we have 9 bits available for address.**

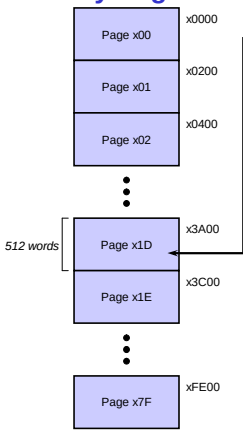
Solution:

- **Upper 7 bits of address are specified (implicitly) by the PC.**

Think of memory as collection of 512-word pages.

- **Upper 7 bits identify which page – the page number.**
- **Lower 9 bits identify which word within the page – the page offset.**

Memory Pages



x3B24
0011101100100100
Page number (7 bits): x1D
Page offset (9 bits): x124

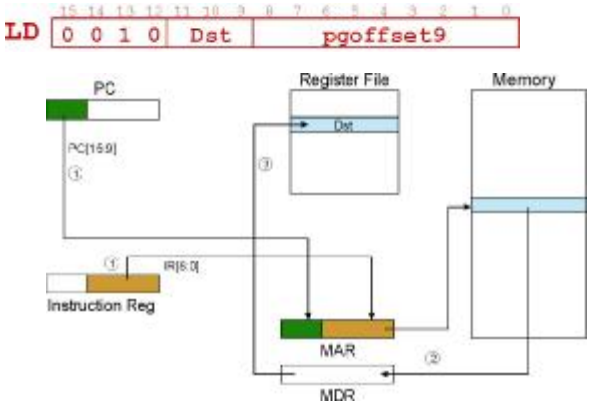
Direct mode addressing gets page number from PC[15:9] and page offset from IR[8:0].

Practice

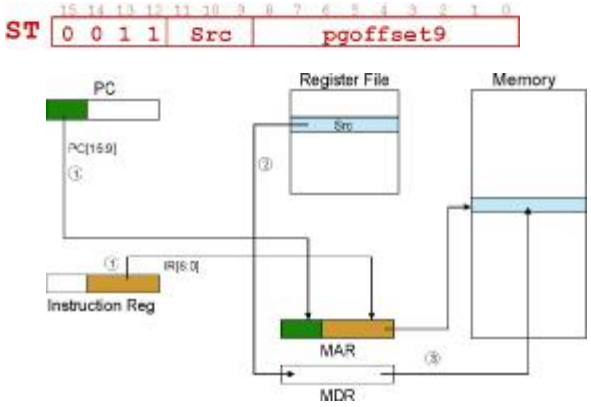
What is the page number and page offset for each of these addresses?

Address	Page Number	Page Offset
x3102		
x3002		
x4321		
xF3FE		

LD (Direct)



ST (Direct)



Base + Offset Addressing Mode

With direct mode, can only address words on the same memory page as the instruction.

- **What about the rest of memory?**

Solution:

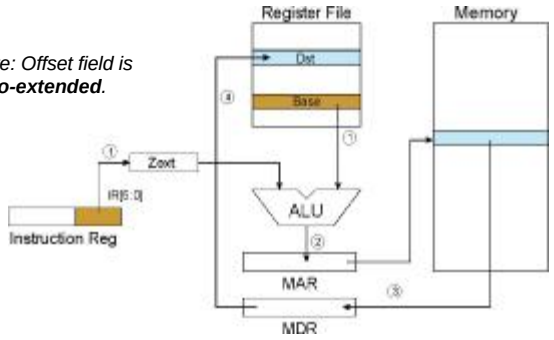
- **Use a register to generate a full 16-bit address.**

4 bits for opcode, 3 for src/dest register, 3 bits for *base* register -- remaining 6 bits are used as an unsigned *offset*.

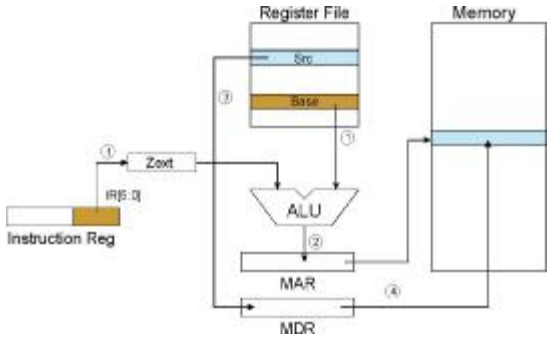
- Offset is *zero-extended* before adding to base register.

LDR (Base+Offset)

Note: Offset field is ***zero-extended****.*



STR (Base+Offset)



Indirect Addressing Mode

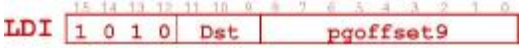
Another way to have a full 16-bit address:

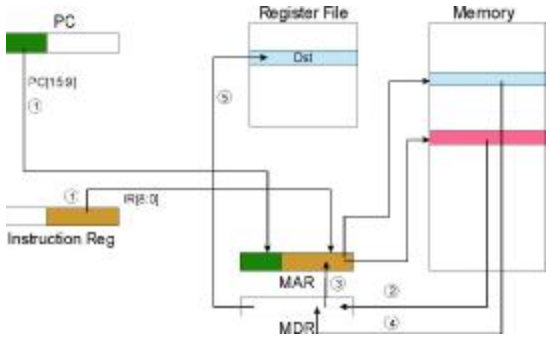
- **Read address from memory location, then load/store to that address.**

First address is generated from PC and IR (just like direct addressing), then content of that address is used as target for load/store.

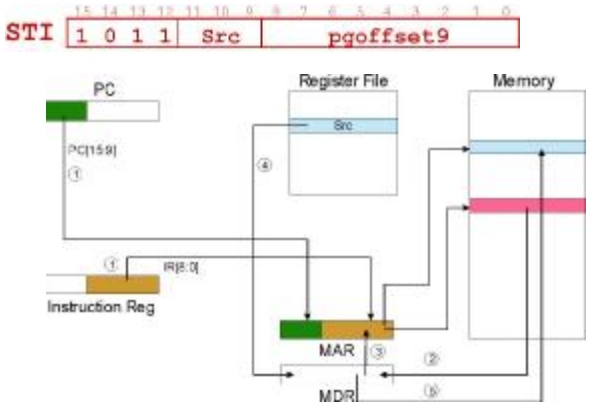
- **Advantage: Doesn't consume a register for base address.**
- **Disadvantage: Extra memory operation (and no offset).**

LDI (Indirect)





STI (Indirect)

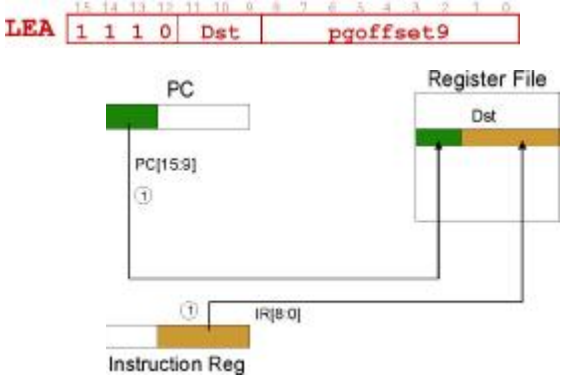


Load Effective Address

Concatenates current page number (PC[15:9]) with page offset (IR[8:0]), and stores the result into a register.

Note: The *address* is stored in the register, not the contents of the memory location.

LEA (Immediate)



Example

Address	*Instruction*	*Comments*
x30F6	1 1 1 0 0 0 1 0 1 1 1 1 0 1 0 0	*R1 ¬ x30F4*
x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	*R2 ¬ R1 + 14 = x3102*
x30F8	0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 0	*M[x30F4] ¬ R2*
x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ¬ 0*
x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	*R2 ¬ R2 + 5 = 5*
x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	*M[R1+14] ¬ R2* *M[x3102] ¬ 5*
x30FC	1 0 1 0 0 1 1 0 1 1 1 1 0 1 0 0	*R3 ¬ M[M[x30F4]]* *R3 ¬ M[x3102]* *R3 ¬ 5*

opcode

Control Instructions

Used to alter the sequence of instructions (by changing the Program Counter)

Conditional Branch

- branch is *taken* if a specified condition is true
  - Ø offset is concatenated with upper bits of PC to yield new PC
- else, the branch is *not taken*
  - Ø PC is not changed, points to the next sequential instruction

Unconditional Branch (or Jump)

- always changes the PC

TRAP

- changes PC to the first instruction in an OS "service routine"
- when routine is done, will execute next instruction

Condition Codes

LC-2 has three condition code registers:

- **N -- negative**
- **Z -- zero**
- **P -- positive (greater than zero)**

Set by any instruction that stores a value to a register (ADD, AND, NOT, LD, LDR, LDI, LEA)

Exactly one will be set at all times

- Based on the last instruction that altered a register

Branch Instruction

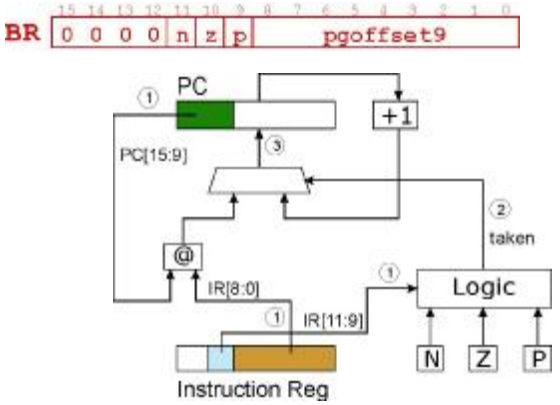
Branch specifies one or more condition codes

If the set bit is specified, the branch is taken

- PC is set to the address specified in the instruction
- Like direct mode addressing, target address is made by concatenating current page number (PC[15:9]) with offset (IR[8:0])
- Note: Target must be on same page as BR instruction.

If the branch is not taken, the next sequential instruction (PC+1) is executed.

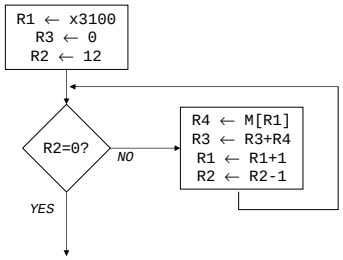
BR



What happens if bits [11:9] are all zero? All one?

Using Branch Instructions

Compute sum of 12 integers.
Numbers start at location x3100. Program starts at location x3000.



Sample Program

Address	*Instruction*	*Comments*
x3000	1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0	*R1 ¬ x3100*
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	*R3 ¬ 0*
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ¬ 0*
x3003	0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 0	*R2 ¬ 12*
x3004	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1	*If Z, goto x3009*
x3005	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	*Load next value to R4*
x3006	0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1	*Add to R3*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*Increment R1 (pointer)*
X3008	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	*Decrement R2 (counter)*
x3009	0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0	*Goto x3004*

Jump Instructions

Jump is an unconditional branch -- *always* taken.

Direct

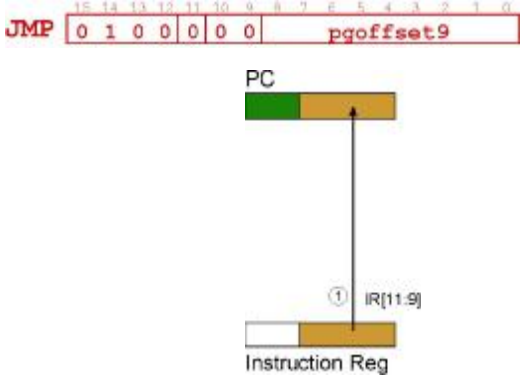
- **Concatenate page number (PC[15:9]) and offset (IR[8:0]).**
- **Works if target is on same page.**

Base + Offset

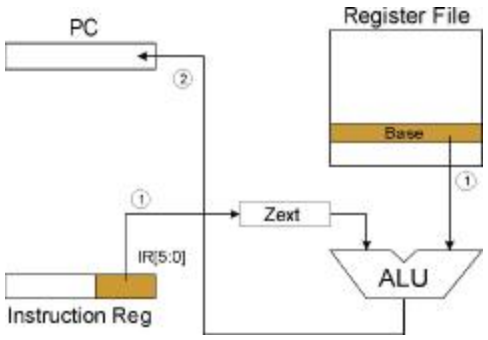
- **Address is register plus unsigned offset (IR[5:0]).**
- **Allows any target address.**

Link bit converts JMP to JSR (Jump to Subroutine).
Will discuss later.

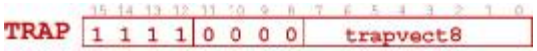
JMP (Direct)



JMPR (Base + Offset)



TRAP



Calls a service routine, identified by 8-bit "trap vector."

vector	*routine*
x23	input a character from the keyboard
x21	output a character to the monitor
x25	halt the program

When routine is done,
PC is set to the instruction following TRAP.
(We'll talk about how this works later.)

Another Example

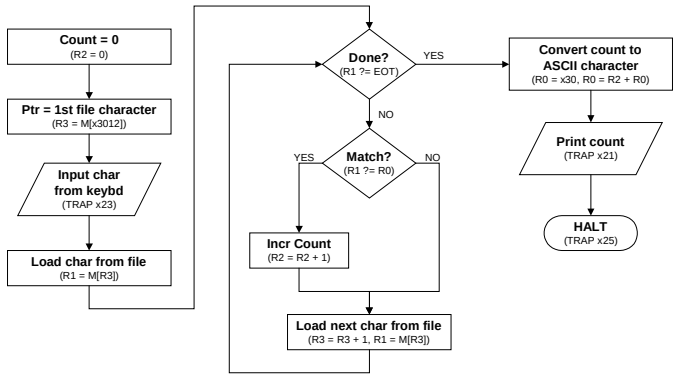
Count the occurrences of a character in a file

- **Program begins at location x3000**
- **Read character from keyboard**
- **Load each character from a "file"**
  - Ø File is a sequence of memory locations
  - Ø Starting address of file is stored in the memory location immediately after the program
- **If file character equals input character, increment counter**
- **End of file is indicated by a special ASCII value: EOT (x04)**
- **At the end, print the number of characters and halt**
  (assume there will be less than 10 occurrences of the character)

A special character used to indicate the end of a sequence is often called a sentinel.

- Useful when you don't know ahead of time how many times to execute a loop.

Flow Chart



Program (1 of 2)

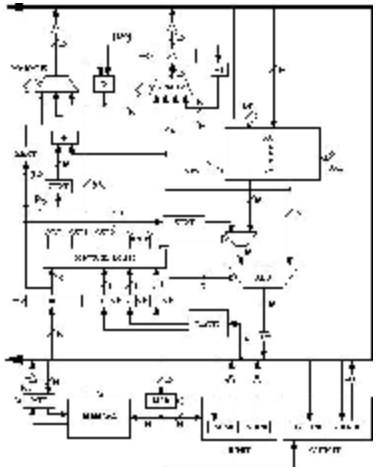
Address	Instruction	Comments
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ¬ 0 (counter)*
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0	*R3 ¬ M[x3102] (ptr)*
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	*Input to R0 (TRAP x23)*
x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ¬ M[R3]*
x3004	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0	*R4 ¬ R1 – 4 (EOT)*
x3005	0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0	*If Z, goto x300E*
x3006	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	*R1 ¬ NOT R1*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*R1 ¬ R1 + 1*
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	*R1 ¬ R1 + R0*
x3009	0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1	*If N or P, goto x300B*

Program (2 of 2)

Address	Instruction	Comments
x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	*R2 ¬ R2 + 1*
x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	*R3 ¬ R3 + 1*
x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ¬ M[R3]*
x300D	0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0	*Goto x3004*
x300E	0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1	*R0 ¬ M[x3013]*
x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0	*R0 ¬ R0 + R2*
x3010	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1	*Print R0 (TRAP x21)*
x3011	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	*HALT (TRAP x25)*
X3012	Starting Address of File	
x3013	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	*ASCII x30 ('0')*

LC-2 Data Path Revisited

Filled arrow
= info to be processed.
Unfilled arrow
= control signal.



Data Path Components

Global bus

- special set of wires that carry a 16-bit signal to many components
- inputs to the bus are "tri-state devices," that only place a signal on the bus when they are enabled
- only one (16-bit) signal should be enabled at any time
  - Ø control unit decides which signal "drives" the bus
- any number of components can read the bus
  - Ø register only captures bus data if it is write-enabled by the control unit

Memory and I/O

- Control and data registers for memory and I/O devices
- memory: MAR, MDR (also control signal for read/write)
- input (keyboard): KBSR, KBDR
- output (monitor): CRTSR, CRTDR

Data Path Components

ALU

- Accepts inputs from register file and from sign-extended bits from IR (immediate field).
- Output goes to bus.
  - Ø used by condition code logic, register file, memory and I/O registers

Register File

- Two read addresses, one write address
- Input from bus
  - Ø result of ALU operation or memory (or I/O) read
- Two 16-bit outputs
  - Ø used by ALU, PC, memory address
  - Ø data for store instructions passes through ALU

Data Path Components

PC and PCMUX

- Four inputs to PC, controlled by PCMUX
  1. current PC plus 1 -- normal operation
  2. PC[15:9] and IR[8:0] -- BR instruction (and JSR, discussed later)
  3. register file -- RET instruction (discussed later)
  4. bus -- TRAP, JSRR instructions (discussed later)

MAR and MARMUX

- Three inputs to MAR, controlled by MARMUX
  1. PC[15:9] and IR[8:0] -- direct addressing mode
  2. Register File plus zero-extended offset -- base+offset mode
  3. Zero-extended IR[7:0] -- TRAP instruction (discussed later)

Data Path Components

Condition Code Logic

- Looks at value on bus and generates N, Z, P signals
- Registers set only when control unit enables them
  - Ø only certain instructions set the codes
    (anything that loads a value into a register:
    ADD, AND, NOT, LD, LDI, LDR, LEA)

Control Unit

- Decodes instruction (in IR)
- On each machine cycle, changes control signals for next phase of instruction processing
  - Ø who drives the bus?
  - Ø which registers are write enabled?
  - Ø which operation should ALU perform?
  - Ø …