

# LAB-4 DISCUSSION

FALL 2010, CSE 140L, UCSD

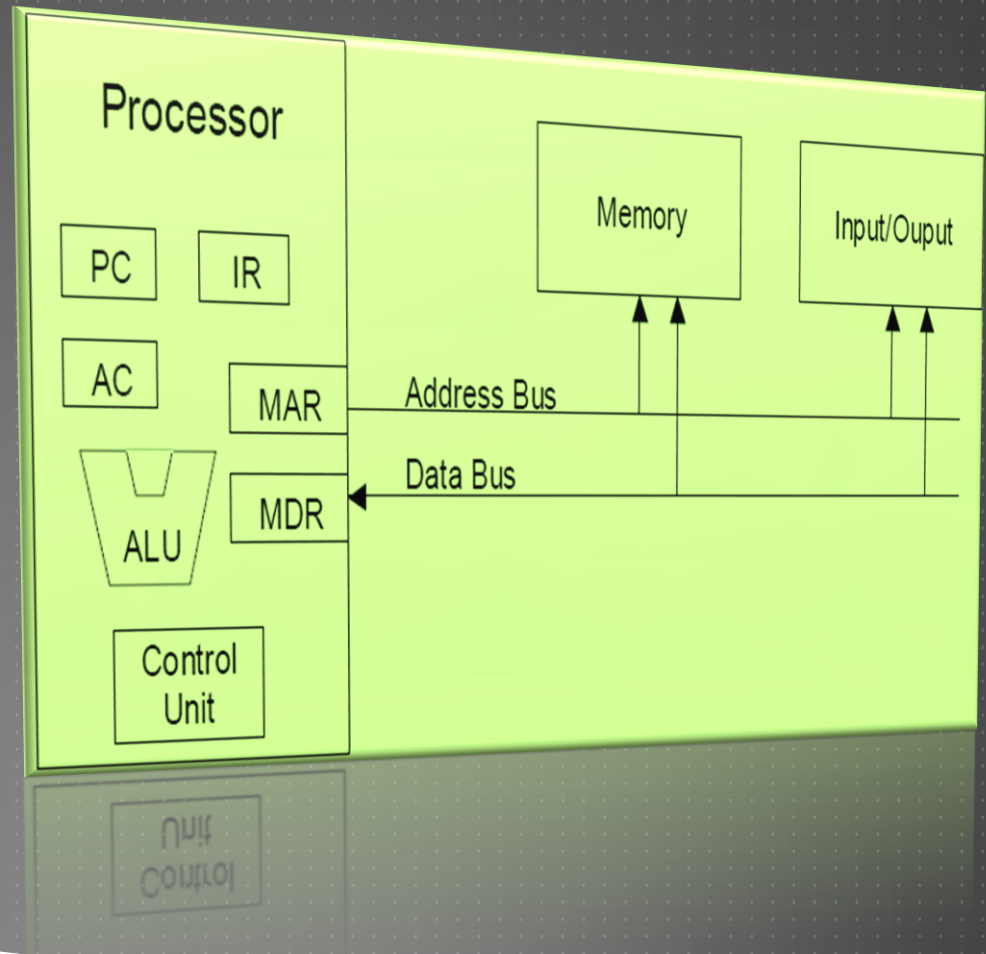
GOPITUMMALA

# OBJECTIVE

- ▶ In this lab, we will build a tiny computer **TC140L - Tiny Computer I40Lab** - (description given separately) in Verilog. The execution results will be displayed in the LED digits of your board. Unlike a real computer, our tiny computer will consist of few instructions.

# SYSTEM OVERVIEW

- ▶ A traditional digital computer consists of three main units
  - ▶ The processor or central processing unit (CPU)
    - ▶ The memory that stores program instructions and data
    - ▶ The input/output hardware that communicates to other devices
  - ▶ Typically, signals on the bus include the memory address, memory data, and bus status.



# INTERNALS

- ▶ Internally, the CPU contains a small number of registers that are used to store data inside the processor.
- ▶ Registers such as PC, IR, AC, MAR and MDR are built using D flip-flops for data storage. One or more arithmetic logic units (ALUs) are also contained inside the CPU.
- ▶ The ALU is used to perform arithmetic and logical operations on data values.
- ▶ Common ALU operations include add, subtract, multiplication and logical and/or operations.

# COMPUTER PROGRAMS AND INSTRUCTIONS

- ▶ A computer program is a sequence of instructions that perform a desired operation. Instructions are stored in memory.
- ▶ in this Lab-4, an instruction consists of 16 bits.



- ▶ The high eight bits of the instruction contain the instruction operation code or "opcode" specifying the operation, such as add
- ▶ The low eight bits of each instruction contain a memory address field. Depending on the opcode, this address may point to a data location or the location of another instruction.

# SMALL ISA AND EXAMPLE

Instruction	Mnemonic	Operation Performed	Opcode Value
ADD	address	$AC \leq AC + \text{contents of memory address}$	00
STORE	Address	$\text{contents of memory address} \leq AC$	01
LOAD	Address	$AC \leq \text{contents of memory address}$	02
JUMP	Address	$PC \leq \text{address}$	03

Example Computer Program for  $A = B + C$ :

Assembly Language

Machine Language

LOAD B

0211

ADD C

0012

STORE A

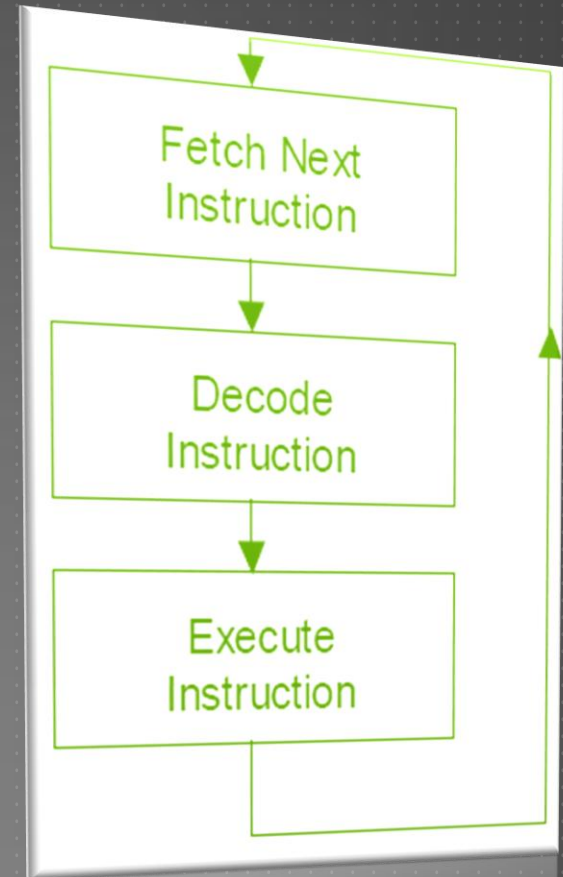
0110

# CONTROL PART

- ▶ The primary operation performed by the processor is the execution of sequences of instructions stored in main memory.
- ▶ The CPU (processor) fetches (reads) an instruction from memory, decodes the instruction to determine what operations are required, and then executes the instruction.
- ▶ The control unit controls this sequence of operations in the processor

# CONTROL FLOW AND PATH

- ▶ A simple state machine called the control unit controls the sequence of operations in the processor
- ▶ The CPU contains a
  - ▶ General-purpose data register called the accumulator (AC)
  - ▶ The program counter (PC)
  - ▶ The arithmetic logic unit (ALU) - used for arithmetic and logical operations
- ▶ The processor fetches an instruction from memory, decodes the instruction to determine what operations are required, and then executes the instruction





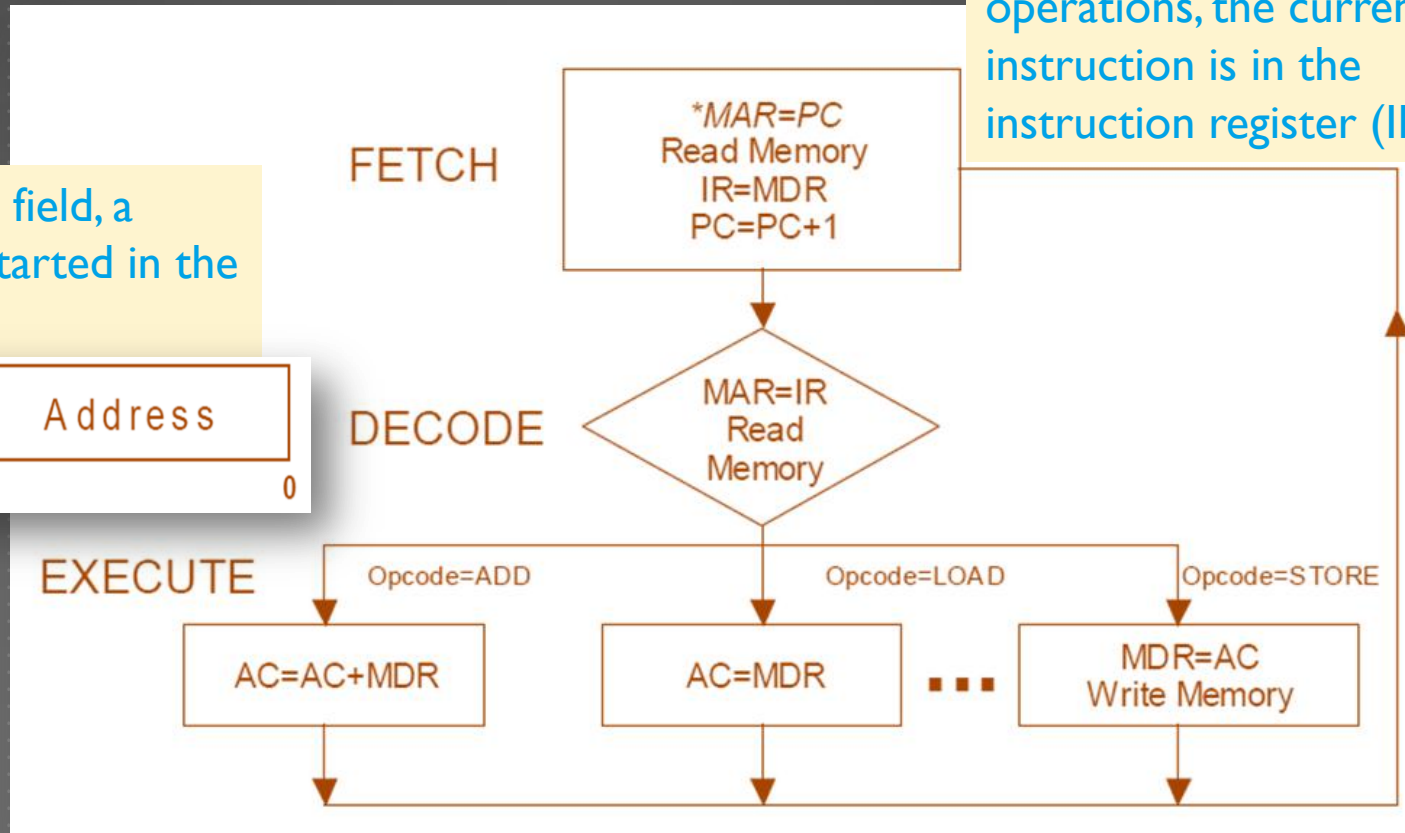
# REGISTER TRANSFER OPERATIONS

- ▶ The program counter contains the address of the current instruction.
- ▶ To fetch the next instruction from memory the processor must increment the program counter (PC).
- ▶ The processor must then send the address value in the PC to memory over the bus by loading the memory address register (MAR) and start a memory read operation on the bus.
- ▶ After a small delay, the instruction data will appear on the memory data bus lines, and it will be latched into the memory data register (MDR).
- ▶ Execution of the instruction may require an additional memory cycle so the instruction is normally saved in the CPU's instruction register (IR).
- ▶ Using the value in the IR, the instruction can now be decoded.
- ▶ Execution of the instruction will require additional operations in the CPU and perhaps additional memory operations.
- ▶ The Accumulator (AC) is the primary register used to perform data calculations and to hold temporary program data in the processor.
- ▶ After completing execution of the instruction the processor begins the cycle again by fetching the next instruction

# CONTROL UNIT FSM

Using the address field, a memory read is started in the decode state.

After this sequence of operations, the current instruction is in the instruction register (IR).

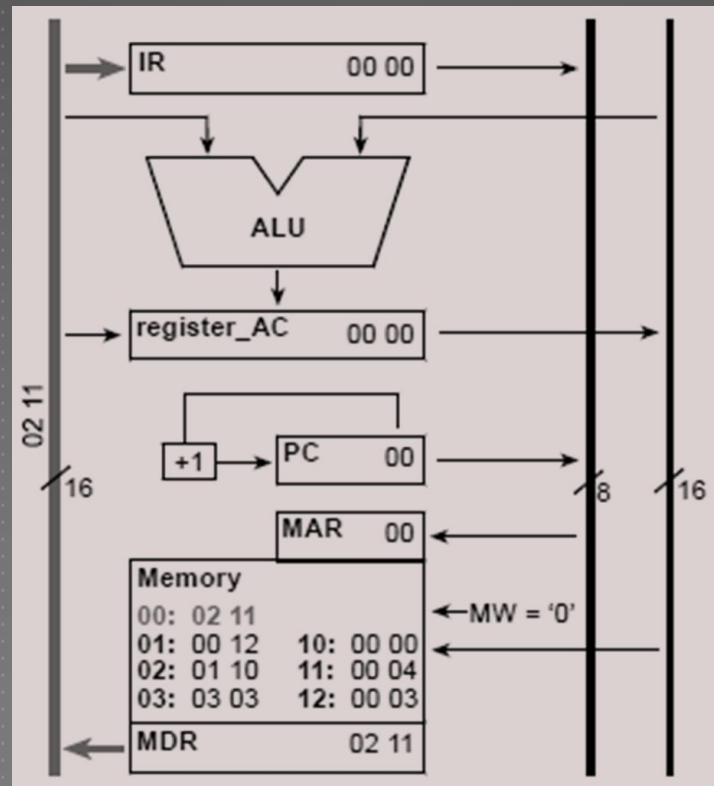


Each instruction requires a short sequence of register transfer operations to implement or execute that instruction

# DATAPATH

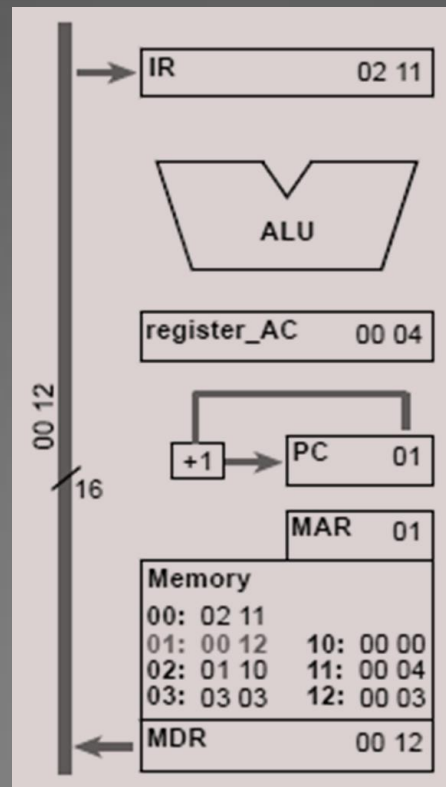
- ▶ A computer's datapath consists of the registers, memory interface, ALUs, and the bus structures used to connect them.
- ▶ The vertical lines are the three major busses used to connect the registers.
- ▶ On the bus lines in the datapath, a "/" with a number indicates the number of bits on the bus.
- ▶ Data values present on the active busses are shown in hexadecimal.
- ▶ MW is the memory write control line.
- ▶ Since the PC and MAR are reset to 00, program execution will start at 00.

# DATAPATH - DESIGN AFTER APPLYING RESET



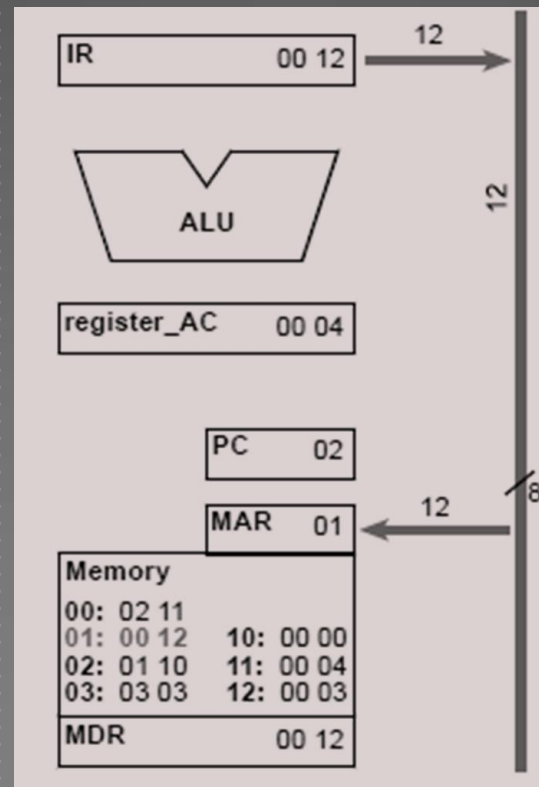
# DATAPATH – FETCH STATE

1. MAR = PC prior to fetch,
2. Read memory,
3. IR = MDR,
4. PC = PC + I



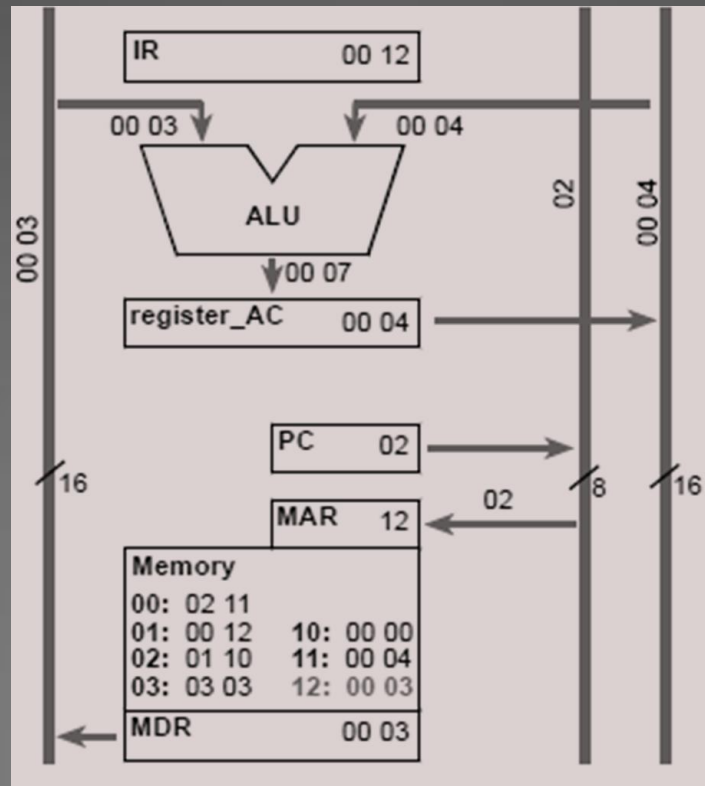
# DATAPATH – DECODE STATE

1. Decode Opcode to find Next State,
2. MAR = IR, and
3. Start memory read



# DATAPATH – EXECUTE STATE

1.  $AC = AC + MDR$ ,
2.  $MAR = PC^*$ , and
3. GOTO FETCH



# VERILOG CODE

1. To demonstrate the operation of the tiny computer using Verilog, a Verilog model of the tiny computer is given
2. The computer's RAM memory is implemented using the Altsyncram function which uses the FPGA's internal memory blocks.
3. The remainder of the computer model is basically a Verilog based state machine that implements the fetch, decode, and execute cycle.
4. The first few lines declare internal registers for the processor along with the states needed for the fetch, decode and execute cycle.
5. A long CASE statement is used to implement the control unit state machine. A reset state is needed to initialize the processor.
6. In the reset state, several of the registers are reset to zero and a memory read of the first instruction is started.
7. This forces the processor to start executing instructions at location 00 in a predictable state after a reset.
8. A second case statement at the end of the code makes assignments to the memory address register based on the current state.



# QUESTIONS – PART I

- ▶ Full computer implementation for ADD instruction is given and you should extend this code for rest of the instructions. Most of the modules and structure of the code is provided. You need to extend the given code and implement the following:
  1. Instruction Fetch Stage
  2. Instruction Decoder Stage (for additional instructions)
  3. Control/Execute FSM (sequential)
  4. ALU (combinational) for the instructions

# INSTRUCTION SET ARCHITECTURE

Instruction	Mnemonic	Operation Performed	Opcode Value
ADD	Address	$AC \leq AC + \text{contents of memory address}$	00
STORE	Address	$\text{contents of memory address} \leq AC$	01
LOAD	Address	$AC \leq \text{contents of memory address}$	02
JUMP	Address	$PC \leq \text{address}$	03
JNEG	Address	If $AC < 0$ Then $PC \leq \text{address}$	04
SUB	Address	$AC = AC - MDR$	05
XOR	Address	$AC = AC \text{ XOR } MDR$	06
OR	Address	$AC = AC \text{ OR } MDR$	07
AND	Address	$AC = AC \text{ AND } MDR$	08
JPOS	Address	If $AC > 0$ THEN $PC \leq \text{address}$	09
JZERO	Address	If $AC = 0$ Then $PC \leq \text{address}$	0A
ADDI	Data	$AC = AC + \text{Data}$	0B
OUT	xxxx	7-Seg LED displays hex value of AC	0C
SHL	Data	$AC = AC$ shifted left by data bits	0D
SHR	Data	$AC = AC$ shifted right by data bits	0E

# SPECIAL INSTRUCTIONS – PART I

1. In the logical XOR instruction each bit is exclusive OR'ed with the corresponding bit in each operation for a total of sixteen independent exclusive OR operations.
2. This is called a bitwise logical operation.
3. OR and AND are also bitwise logical operations.
4. For Shift instructions, only the low four bits of the address field contain the shift amount. The other four bits are always zero.
5. For OUT instructions modify or use only the low eight bits of AC.

# QUESTIONS – PART II

- ▶ Find the maximum clock rate of the Tiny computer.
- ▶ Examine the project's compiler report and find the logic cell (LC) percentage utilized
- ▶ Do the compilation only for TCI40L module

# QUESTIONS – PART II

The TCI40L's multiple clock cycles per instruction implementation approach was used in early generation microprocessors. These computers had limited hardware, since the VLSI technology at that time supported orders of magnitude fewer gates on a chip than is now possible in current devices.

- ▶ Google and explain in no more than a paragraph as to what the following are
  - ▶ Pipelined processor
  - ▶ Superscalar processor
- ▶ How could you extend the Tiny computer design to modernize by incorporating the above features. (Write in two/three points)

# VERILOG FILES

- ▶ Clock\_divider.v – Feel free to hack here to change the operating clock
- ▶ HexDigit.v
- ▶ Instruction\_decoder.v – Write your decoding logic here
- ▶ Instruction\_fetch.v – Write your fetch logic here
- ▶ Lab4\_de1.v – Don't Update – Top-level file with DE1 pin-mapping
- ▶ Lab4\_de2.v – Don't Update – Top-level file with DE1 pin-mapping
- ▶ Tc1401.v – Please update FSM inside this file.

# TO LOAD A DIFFERENT TEST PROGRAM

```
// Reads in mif file for initial program and data values  
//RAM.init_file           = "fibol.mif",  
RAM.init_file           = "new_test_program.mif",
```

Comment out the given fibonacci test program and overwrite with your new test program

# CLOCK HACK

- ▶ The clock will can be varied by hacking clock\_divider.v

- ▶ e.g.:

```
parameter DIV_CONST = 10000000; // for 1Hz clock
```

```
parameter DIV_CONST = 4000000; // etc
```

- ▶ Feel free to hack this constant to slow down the clock or accelerate it.



# INPUT KEYS

## Buttons on Board:

*Push/Toggle buttons:*

**KEY[0]**: reset

**SW[3]**: Input clock ticks (used to debug)

**SW[4]**: Selects between internal clock and your Inputs to tick the processor from KEY[3] i.e. Pushbutton 3

# OUTPUT KEYS

**OUTPUT:** Four numbers output to the 4 LED displays in hexadecimal format

**Buttons on Board:**

*Toggle Keys:*

**SW[2:0]:** Selects between the following displays

3'b000: Accumulator

3'b001: PC

3'b010: MDR (Memory Data Register)

3'b011: IR

3'b100: OUT

Any questions ?

@WEBCT