

CPS 104
Computer Organization
MYMIPS:
ISA, Simulator

Robert Wagner

Overview of Today's Lecture:

- Exam Reminder
- Exam Sample
- From software to hardware
- Clear distinctions: Hardware simulator,
Loader, Assembler
- Simulator Design
- Assembler Design

Exam Reminder

- **In Class Mid-term Exam:**
- **Bring SPIM manuals, notes, homework problems.**

Sample Exam Questions

- **What is the 6-bit 2's complement representation of -5?**
 - **What is the decimal value of -5, shifted right 1 bit arithmetically?**
 - **What is the decimal value of $5 \gg 1$?**
 - **How do you tell if a negative 2's complement number is odd?**
- **Translate the following C program into equivalent SPIM:**

```
int l=1,j=0,k=10; tbl[10]={1,4,10,5,0,6};  
while (l && j<k) {j++; l=tbl[j]; }
```
- **SPIM compiles pseudo-instructions into 1-3 MIPS instructions**
 - **What sequence of bare machine instructions can SPIM use for:**
 - **la \$3,0x1001000**
 - **lw \$3,0x1008000**
 - **lw \$3,0x1008000(\$4)**
 - **lw \$3,0x100(\$4)**

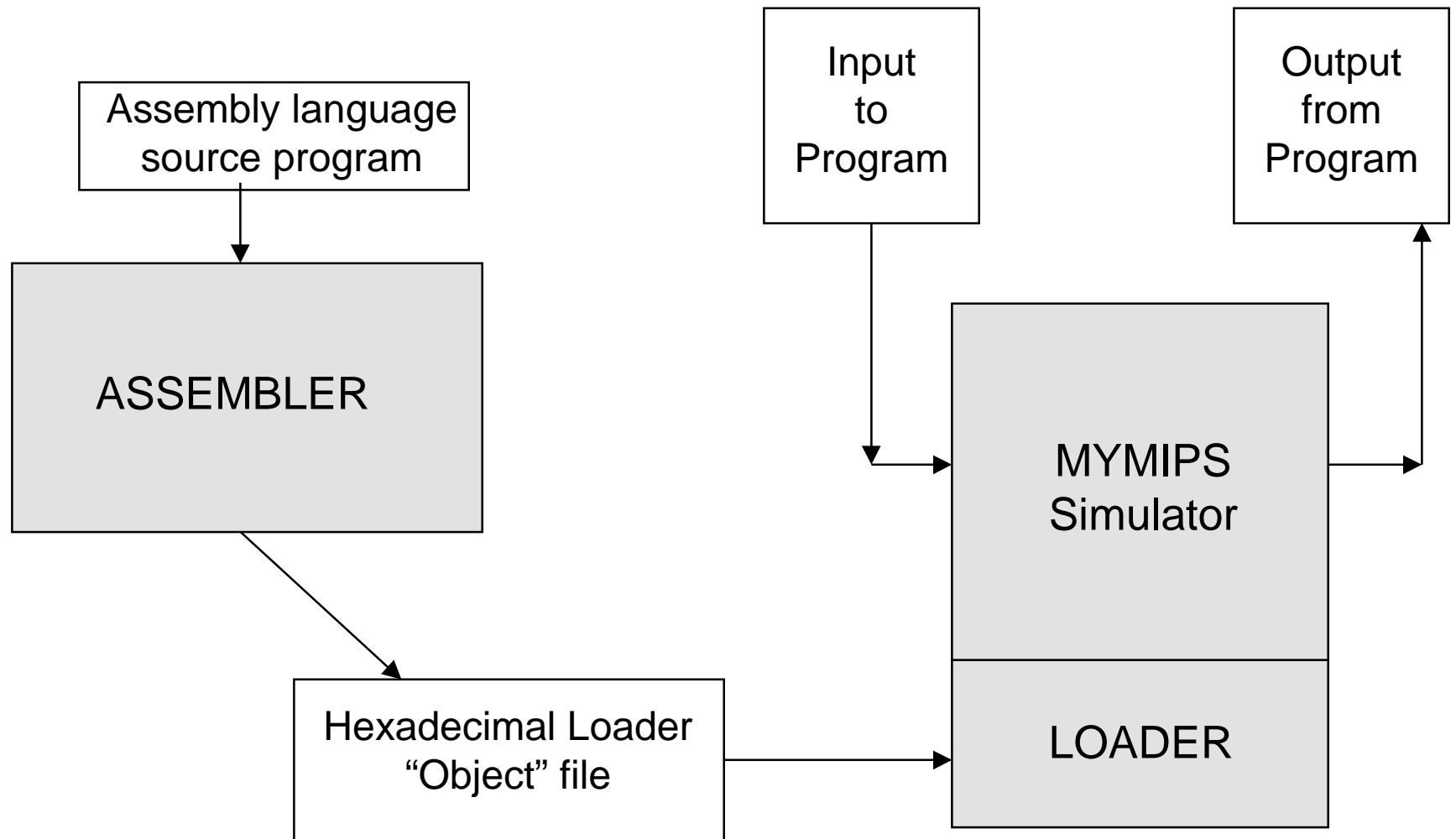
Exam Example: Error conditions in MYMIPS: Occur? How detect?:

- a. **Address outside memory boundaries**
- b. **Address not aligned**
- c. **Branch to data segment**
- d. **Illegal operation code**
- e. **Illegal branch condition**
- f. **Illegal SYSCALL**
- g. **Illegal destination register number**
- h. **Illegal source register number**
- i. **Immediate constant too large**
- j. **Result of addition outside representable range**
- k. **Program never stops**

Software Hardware Interface

- **SPIM emphasizes programming ease, conceals hardware complexity**
- **MYMIPS simulator emphasizes how hardware functions logically (high level)**
- **Lower level hardware functioning: Later in course**

Overview of pieces



Assembler Operation

- **Maintains “hash lookup table” to associate NAMES with “definitions”:**
 - **Instruction Op-code Values + Formats (pre-defined)**
 - **Assembler Directive Actions (pre-defined)**
 - **Memory Locations (for statement labels)**
- **Uses 2 “passes” over source:**
 - **Pass 1: Decide where each assembled word will be located in memory (# words each source line represents)**
 - **Define each label equal to location of item it labels**
 - **Pass 2: Use these definitions to build Hex representation of each word (fill instruction fields with proper values from the hash table “value” associated with each name)**

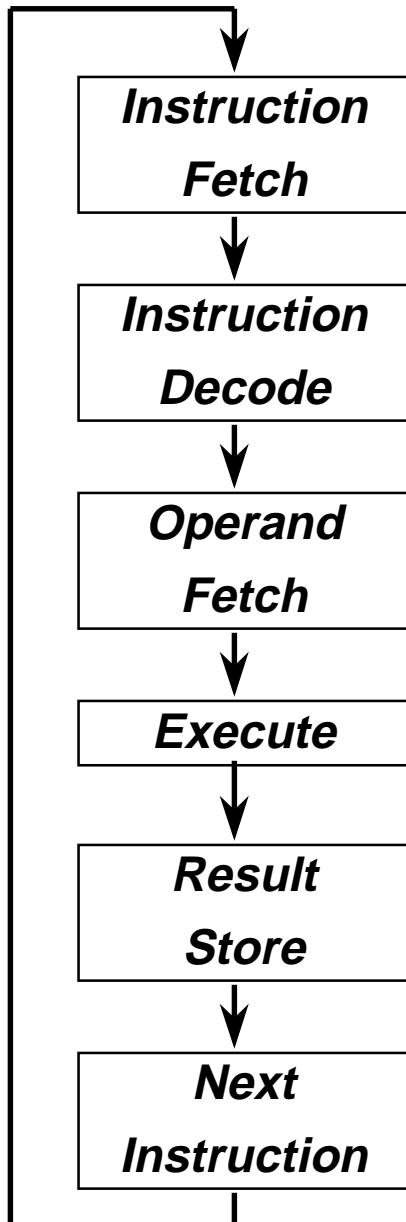
Loader

- **Simple: Convert Hexadecimal constants into internal binary words**
 - Suggest using “fgets()” to read lines, then “sscanf()” to convert numbers on the line read in from Hex to internal binary
 - sscanf() reads from a STRING, not stdin
 - sscanf() returns # of #s it found -- use to identify END
 - Format code %x does Hex conversion (uses lower-case a-f)
- When “-1 n” loader directive found, start Simulator with
 - PC=MEMORY[n]. n is a BYTE address.
- May have to “fool” C to allow both BYTE and WORD access to MEMORY

```
char MEMORY[1<<19];  
*((int *) &MEMORY[b]) = word;
```

Should place “word” into bytes MEMORY[b], ..., MEMORY[b+3].
“b” MUST be a multiple of 4 for this to work

MYMIPS Simulator



- Get instruction from MEMORY[PC]
- Split instruction fields apart
 - Use “shift and mask” : e.g. $(inst \gg 6) \& 0xFF$
- Get register operands + compute $e()$ and $eu()$ into scalar variables
- Use switch to select action from Opcode value and perform selected action
 - (This is part of “Execute” box)
 - (This is part of “Execute” box)