CPT_S 260 Intro to Computer Architecture
Lecture 13

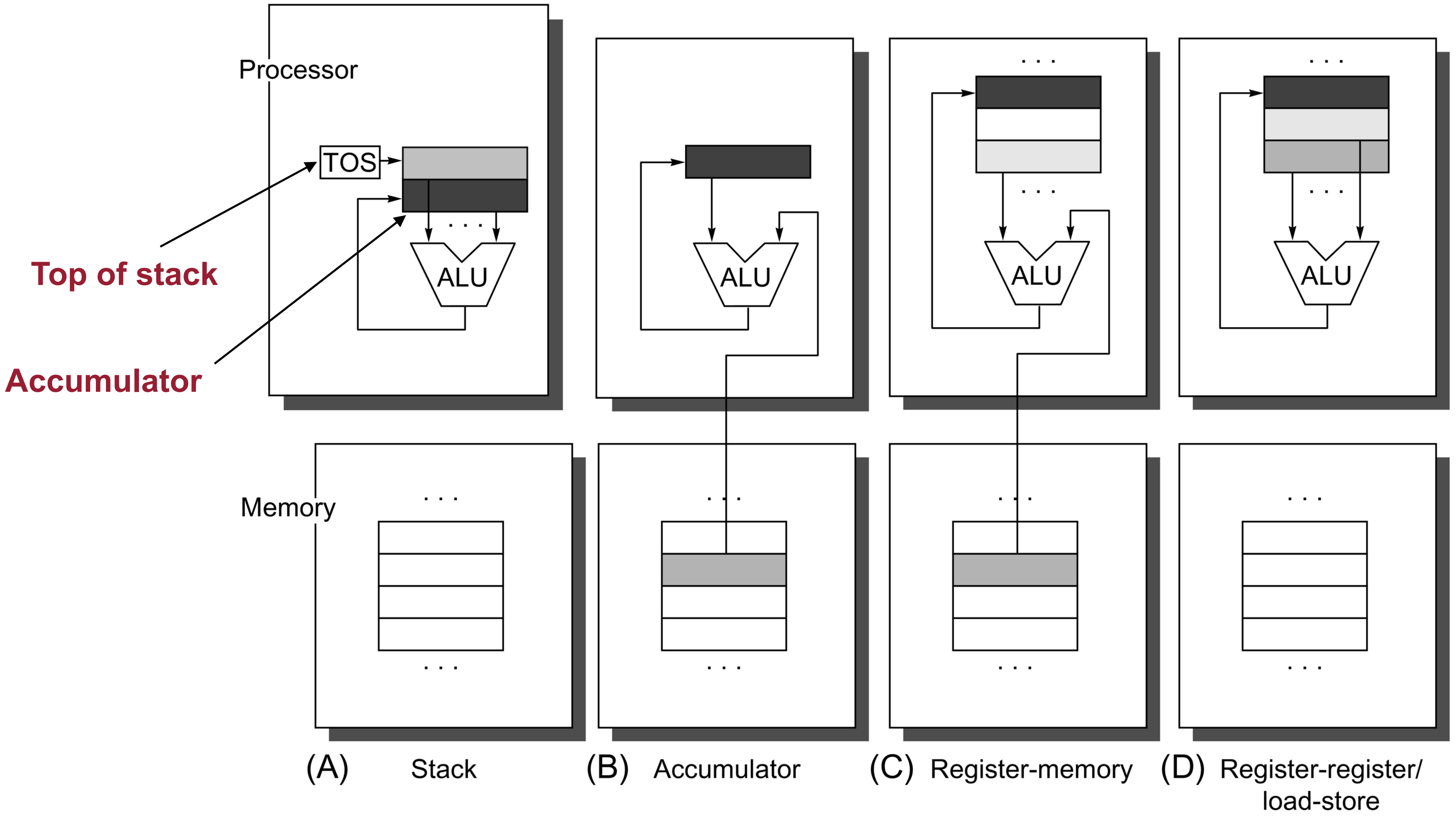
Intro to MIPS II
February 9, 2022

Ganapati Bhat

School of Electrical Engineering and Computer Science

Washington State University

Recap: ISA Classification Visualization



Recap: Issues in Instruction Set Design

- Memory addressing
- Type and size of operands
- Operations in the instruction set
- Encoding of ISAs
- Implementation (pipelining, scheduling, etc.)

Recap: MIPS Instruction Goals

- **Showing how it is represented in hardware**
- **The relationship between high-level programming languages and this more primitive one**
  - Examples in C and JAVA
- **See the impact of programming languages and compiler optimization on performance**
- **Writing programs in the language of the computer and running them on the simulator**

Recap: Arithmetic Operations

- **Add and subtract, three operands**
  - Two sources and one destination

```
add a, b, c  # a gets b + c
```

- **All arithmetic operations have this form**
- ***Design Principle 1:* Simplicity favors regularity**
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Register Operands

- **Arithmetic instructions use register operands**
- **MIPS has a 32 × 32-bit register file**
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a “word”
- **Assembler names**
  - \$t0, \$t1, …, \$t9 for temporary values
  - \$s0, \$s1, …, \$s7 for saved variables
- ***Design Principle 2:* Smaller is faster**
  - c.f. main memory: millions of locations

Register Operand Example

- **C code:**

```
f = (g + h) - (i + j);
```

For instance, f, …, j stored in $s0, …, $s4

- **Compiled MIPS code:**

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Data Types

- Instructions are all 32 bits
- Byte(8 bits), halfword (2 bytes), word (4 bytes)
- A character requires 1 byte of storage
- An integer requires 1 word (4 bytes) of storage

Memory Operands

- **Main memory used for composite data**
  - Arrays, structures, dynamic data
- **To apply arithmetic operations**
  - Load values from memory into registers
  - Store result from register to memory
- **Memory is byte addressed**
  - Each address identifies an 8-bit byte
- **Words are aligned in memory**
  - Address must be a multiple of 4
- **MIPS is Big Endian**
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address

Memory Operand Example #1

- **C code:**

```
g = h + A[8];
```

  – Assume that g is in \$s1, h in \$s2, base address of A in \$s3

- **Compiled MIPS code:**
  – Index 8 requires offset of 32
    » 4 bytes per word

```
lw  $t0, 32($s3)    # load word
add $s1, $s2, $t0
```

offset **base register**

Memory Operand Example #2

- **C code:**

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3

- **Compiled MIPS code:**
  - Index 8 requires offset of 32

```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```

Registers vs. Memory

- **Registers are faster to access than memory**
- **Operating on memory data requires loads and stores**
  - More instructions to be executed
- **Compiler must use registers for variables as much as possible**
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Immediate Operands

- **Constant data specified in an instruction**

```
addi $s3, $s3, 4
```

- **No subtract immediate instruction**
  - Just use a negative constant

```
addi $s2, $s1, -1
```

- ***Design Principle 3:* Make the common case fast**
  - Small constants are common
  - Immediate operand avoids a load instruction

The Constant Zero

- **MIPS register 0 ($zero) is the constant 0**
  - Cannot be overwritten
- **Useful for common operations**
  - E.g., move between registers

```
add $t2, $s1, $zero
```

Register Number	Alternative Name	Description
0	zero	the value 0
1	$at	(**a**ssembler **t**emporary) reserved by the assembler
2-3	$v0 - $v1	(**v**alues) from expression evaluation and function results
4-7	$a0 - $a3	(**a**rguments) First four parameters for subroutine. Not preserved across procedure calls
8-15	$t0 - $t7	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. Not preserved across procedure calls
16-23	$s0 - $s7	(**s**aved values) - Callee saved. A subroutine using one of these must save original and restore it before exiting. Preserved across procedure calls
24-25	$t8 - $t9	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. These are in addition to $t0 - $t7 above. Not preserved across procedure calls.
26-27	$k0 - $k1	reserved for use by the interrupt/trap handler
28	$gp	**g**lobal **p**ointer. Points to the middle of the 64K block of memory in the static data segment.
29	$sp	**s**tack **p**ointer Points to last location on the stack.
30	$s8/$fp	saved value / **f**rame **p**ointer Preserved across procedure calls
31	$ra	**r**eturn **a**ddress

Sign Extension

- **Representing a number using more bits**
  - Preserve the numeric value
- **In MIPS instruction set**
  - `addi`: extend immediate value
  - `lb`, `lh`: extend loaded byte/halfword
  - `beq`, `bne`: extend the displacement
- **Replicate the sign bit to the left**
  - Unsigned values: extend with 0s
- **Examples: 8-bit to 16-bit**
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

Logical Operations

- **Instructions for bitwise manipulation**

Operation	C	Java	MIPS
Shift left	<<	<<	`sll`
Shift right	>>	>>>	`srl`
Bitwise AND	&	&	`and, andi`
Bitwise OR	|	|	`or, ori`
Bitwise NOT	~	~	`nor`

- **Useful for extracting and inserting groups of bits in a word**

AND Operations

- **Useful to mask bits in a word**
  - Select some bits, clear others to 0

```
and $t0, $t1, $t2
```

$t2	**0000 0000 0000 0000 0000 1101 1100 0000**
$t1	**0000 0000 0000 0000 0011 1100 0000 0000**
$t0	**0000 0000 0000 0000 0000 1100 0000 0000**

OR Operations

- **Useful to include bits in a word**
  - Set some bits to 1, leave others unchanged

```
or $t0, $t1, $t2
```

$t2	**0000 0000 0000 0000 0000 1101 1100 0000**
$t1	**0000 0000 0000 0000 0011 1100 0000 0000**
$t0	**0000 0000 0000 0000 0011 1101 1100 0000**

NOT Operations

- **Useful to invert bits in a word**
  - Change 0 to 1, and 1 to 0
- **MIPS has NOR 3-operand instruction**
  - a NOR b == NOT (a OR b)

```
nor $t0, $t1, $zero
```

Register 0: always read as zero

$t1	**0000 0000 0000 0000 0011 1100 0000 0000**
$zero	**0000 0000 0000 0000 0000 0000 0000 0000**
$t0	**1111 1111 1111 1111 1100 0011 1111 1111**