Instruction Set Architecture

Prof. Usagi
Recap: von Neumann Architecture

By loading different programs into memory, your computer can perform different functions.
Recap: How my “C code” becomes a “program”
Recap: How my “Java code” becomes a “program”
Recap: How my “Python code” becomes a “program"
Recap: Demo

```
if (option)
    std::sort(data, data + arraySize);

O(nlog₂n)

for (unsigned c = 0; c < arraySize*1000; ++c) {
    if (data[c%arraySize] >= INT_MAX/2)
        sum ++;
}

O(n)

if option is set to 1:  O(nlog₂n)

otherwise, O(n):  O(n)
```
Outline

• What’s an “instruction set architecture (ISA)”
• Overview of MIPS ISA
• How to express a program in MIPS ISA
Instruction Set Architecture (ISA)
Me

You

I expect the lecture to be...
Peer instruction

• Before the lecture — You need to complete the required reading
• During the lecture — I’ll bring in activities to ENGAGE you in exploring your understanding of the material
  • Popup questions
  • Individual thinking — use polls in Zoom to express your opinion
  • Group discussion
    • Breakout rooms based on your residential colleges!
    • Use polls in Zoom to express your group’s opinion
  • Whole-classroom discussion — we would like to hear from you

Read Think Discuss
By how much you know about ISA?

- How many of the following is generally true about ISAs?
  1. Many models of processors can support one ISA
  2. An ISA is unique to one model of processor
  3. Every processor can support multiple ISAs
  4. Each processor manufacturer has its own ISA

A. 0
B. 1
C. 2
D. 3
E. 4
By how much you know about ISA?

• How many of the following is generally true about ISAs?
  ① Many models of processors can support one ISA
  ② An ISA is unique to one model of processor
  ③ Every processor can support multiple ISAs
  ④ Each processor manufacturer has its own ISA

A. 0
B. 1
C. 2
D. 3
E. 4
Popular ISAs
How many of the following is generally true about ISAs?

- Many models of processors can support one ISA
- An ISA is unique to one model of processor
- Every processor can support multiple ISAs
- Each processor manufacturer has its own ISA

A. 0
B. 1
C. 2
D. 3
E. 4
Instruction Set Architecture (ISA)

- The **contract** between the hardware and software
- Provide an **abstraction** of the underlying processor
  - Defines the set of operations that a computer/processor can execute
  - Defines the **memory** space that a program can use
- Programs are combinations of these instructions
- The hardware implements these instructions in any way it choose.
  - Directly in hardware circuit. e.g. CPU
  - Software virtual machine. e.g. VirtualPC
  - Simulator/Emulator. e.g. DeSmuME
  - Trained monkey with pen and paper

We’re abstracting a von Neumann machine!
Instruction Set Architecture (ISA)

add, sub, mul...

lw, sw...

bne, jal...

Emulator/Virtual machine
The “abstraction”
The “abstraction”
Assembly language

- The human-readable representation of “instructions”/“machine language”
- Has a direct mapping between assembly code and instructions
- Assembly may contain “pseudo instructions” for programmer to use
- Each pseudo instruction still has its own mapping to a set of real machine instructions

```
add $v0, $a1, $a2
```

```
000000 00101 00110 00010 00000 100000
```
Elements in an ISA

- **Operations**
  - Types of operations: Arithmetic/Logical, memory access, control-flow (e.g., branch, function calls)
  - How many operations should the processor support?
- **Operands**
  - Types of operands — register, constant, memory addresses
  - Number of operands — 0, 1, 2, 3 or ?
  - Sizes of operands — byte, 16-bit, 32-bit, 64-bit
- **Memory space**
  - The size of memory that programs can use
  - The addressing of each memory locations
  - The modes to represent those addresses
An Overview of MIPS ISA
The abstracted "MIPS" machine

Registers

$zero $at $v0 $v1 $a0 $a1 $a2 $a3 $t0 $t1 $t2 $t3 $t4 $t5 $t6 $t7 $s0 $s1 $s2 $s3 $s4 $s5 $s6 $s7 $t8 $t9 $k0 $k1 $gp $sp $fp $ra

Program Counter

0x0000000000000004

Memory

0x00000000 0x00000008 0x00000010 0x00000018 0x00000020 0x00000028 0x00000030 0x00000038

ALU

add addi

and andi

ori xori

beq blt jal jr

Byte Addressing — every byte of data/instruction has its own address

Memory

0xFFFFFFFFC0 0xFFFFFFFFC8 0xFFFFFFFFD0 0xFFFFFFFFD8 0xFFFFFFFFE0 0xFFFFFFFFE8 0xFFFFFFF0 0xFFFFFFF8

2^32 bytes

Memory

0x00000000 0x00000008 0x00000010 0x00000018 0x00000020 0x00000028 0x00000030 0x00000038

Memory

0xFFFFFFFFC0 0xFFFFFFFFC8 0xFFFFFFFFD0 0xFFFFFFFFD8 0xFFFFFFFFE0 0xFFFFFFFFE8 0xFFFFFFF0 0xFFFFFFF8
All instructions are **32** bits

**32 32-bit registers**
- All registers are the same
- $zero$ is always 0

**50 opcodes**
- Arithmetic/Logic operations
- Load/store operations
- Branch/jump operations

**3 instruction formats**
- R-type: all operands are registers
- I-type: one of the operands is an immediate value
- J-type: non-conditional, non-relative branches

---

<table>
<thead>
<tr>
<th>Reg. Name</th>
<th>Reg. Num</th>
<th>Usage</th>
<th>Saved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>0</td>
<td>zero</td>
<td>N/A</td>
</tr>
<tr>
<td>$at</td>
<td>1</td>
<td>assembler temporary</td>
<td>no</td>
</tr>
<tr>
<td>$v0-$v1</td>
<td>2-3</td>
<td>return value</td>
<td>no</td>
</tr>
<tr>
<td>$a0-$a3</td>
<td>4-7</td>
<td>arguments</td>
<td>no</td>
</tr>
<tr>
<td>$t0-$t7</td>
<td>8-15</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$s0-$s7</td>
<td>16-23</td>
<td>saved</td>
<td>yes</td>
</tr>
<tr>
<td>$t8-$t9</td>
<td>24-25</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$k0-$k1</td>
<td>26-27</td>
<td>OS kernel</td>
<td>no</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>global pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>stack pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>frame pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>return address</td>
<td>yes</td>
</tr>
</tbody>
</table>
## Subset of MIPS instructions

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Usage</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic</strong></td>
<td>add</td>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>addi $s1, $s2, 20</td>
<td>$s1 = $s2 + 20</td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 − $s3</td>
</tr>
<tr>
<td><strong>Logical</strong></td>
<td>and</td>
<td>and $s1, $s2, $s3</td>
<td>$s1 = $s2 &amp; $s3</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>or $s1, $s2, $s3</td>
<td>$s1 = $s2</td>
</tr>
<tr>
<td></td>
<td>andi</td>
<td>andi $s1, $s2, 20</td>
<td>$s1 = $s2 &amp; 20</td>
</tr>
<tr>
<td></td>
<td>sll</td>
<td>sll $s1, $s2, 10</td>
<td>$s1 = $s2 * 2^10</td>
</tr>
<tr>
<td></td>
<td>srl</td>
<td>srl $s1, $s2, 10</td>
<td>$s1 = $s2 / 2^10</td>
</tr>
<tr>
<td><strong>Data Transfer</strong></td>
<td>lw</td>
<td>lw $s1, 4($s2)</td>
<td>$s1 = mem[$s2+4]</td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>lw $s1, 4($s2)</td>
<td>mem[$s2+4] = $s1</td>
</tr>
<tr>
<td><strong>Branch</strong></td>
<td>beq</td>
<td>beq $s1, $s2, 25</td>
<td>if($s1 == $s2), PC = PC + 100</td>
</tr>
<tr>
<td></td>
<td>bne</td>
<td>bne $s1, $s2, 25</td>
<td>if($s1 != $s2), PC = PC + 100</td>
</tr>
<tr>
<td><strong>Jump</strong></td>
<td>jal</td>
<td>jal 25</td>
<td>$ra = PC + 4, PC = 100</td>
</tr>
<tr>
<td></td>
<td>jr</td>
<td>jr $ra</td>
<td>PC = $ra</td>
</tr>
</tbody>
</table>

The only type of instructions can access memory.
- op $rd, $rs, $rt
  - 3 regs.: add, addu, and, nor, or, sltu, sub, subu
  - 2 regs.: sll, srl
  - 1 reg.: jr
- 1 arithmetic operation, 1 I-memory access
- Example:
    - opcode = 0x0, funct = 0x20
  - sll $t0, $t1, 8: R[8] = R[9] << 8
    - opcode = 0x0, shamt = 0x8, funct = 0x0
l-type

- op $rt, $rs, immediate
  - addi, addiu, andi, beq, bne, ori, slti, sltiu
- op $rt, offset($rs)
  - lw, lbu, lhu, ll, lui, sw, sb, sc, sh
- 1 arithmetic op, 1 I-memory and 1 D-memory access
- Example:
  - lw $s0, 4($s2)
  - add $s2, $s2, $s1
  - lw $s0, 0($s2)

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

only two addressing modes
Data transfer instructions

• The ONLY type of instructions that can interact with memory in MIPS

• Two big categories
  • Load (e.g., lw): copy data from memory to a register
  • Store (e.g., sw): copy data from a register to memory

• Two parts of operands
  • A source or destination register
  • Target memory address = base address + offset
    • Register contains the “base address”
    • Constant as the “offset”
  
  \[ 8(\$s0) = (\text{the content in } \$s0) + 8 \]
We have the following C code:

```c
int i, A[512];
A[5] = 0; // sw $zero, 0($t0)
```

If the `sw` instruction implements `A[5]=0` and `&A[0]` is `32'd1000`. What value should `$t0` contain in decimal?

A. 1020
B. 1023
C. 1005
D. 1005
E. 1000
What address should I store to?

- We have the following C code:
  ```c
  int i, A[512];
  A[5] = 0; // sw $zero, 0($t0)
  ```

if the `sw` instruction implements `A[5] = 0` and `&A[0]` is 32'd1000. What value should `$t0` contain in decimal?

A. 1020  
B. 1023  
C. 1005  
D. 1005  
E. 1000
What address should I store to?

- We have the following C code:
  
  ```
  int i, A[512];
  A[5] = 0; // sw $zero, 0($t0)
  ```

  if the `sw` instruction implements `A[5] = 0` and `&A[0]` is 32'd1000. What value should `$t0` contain in decimal?

  A. 1020
  B. 1023
  C. 1005
  D. 1005
  E. 1000
### I-type (cont.)

<table>
<thead>
<tr>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
</tr>
</tbody>
</table>

- `op $rt, $rs, immediate`
  - addi, addiu, andi, beq, bne, ori, slti, sltiu
- `op $rt, offset($rs)`
  - lw, lbu, lhu, ll, lui, sw, sb, sc, sh
- 1 arithmetic op, 1 I-memory and 1 D-memory access

**Example:**
- `beq $t0, $t1, -40`
  - if (R[8] == R[9]) PC = PC + 4 + 4*(-40)
J-type

<table>
<thead>
<tr>
<th>6 bits</th>
<th>26 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>immediate</td>
</tr>
</tbody>
</table>

- op immediate
  - j, jal
- 1 instruction memory access, 1 arithmetic op
- Example:
  - jal quicksort
    - $R[31] = PC + 4$
    - $PC = quicksort$
Practice

• Translate the C code into assembly:

```c
int A[100];
int sum=0,i;

for(i = 0; i < 100; i++)
{
    sum+=A[i];
}
```

1. Initialization (if i = 0, it must < 100)
2. Load A[i] from memory to register
3. Add the value of A[i] to sum
4. Increase by 1
5. Check if i still < 100

There are many ways to translate the C code. But efficiency may be differ among translations.
Number of instructions being executed?

• For the following C code snippet and it's corresponding MIPS translation on the right hand side, how many operations will the processor execute in total to complete all loop iterations?

```c
for(i = 0; i < 100; i++)
{
    sum+=A[i];
}
```

```mips
and $t0, $t0, $zero #let i = 0
addi $t1, $zero, 100 #temp = 100
LOOP: lw  $t3, 0($s0)     #temp1 = A[i]
    add  $v0, $v0, $t3   #sum += temp1
    addi $s0, $s0, 4     #addr of A[i+1]
    addi $t0, $t0, 1     #i = i+1
    bne  $t1, $t0, LOOP  #if i < 100
```

A. 7
B. 402
C. 502
D. 407
E. 507
Number of instructions being executed?

- For the following C code snippet and its corresponding MIPS translation on the right hand side, how many operations will the processor execute in total to complete all loop iterations?

```c
for(i = 0; i < 100; i++)
{
    sum+=A[i];
}
```

```mips
and $t0, $t0, $zero #let i = 0
addi $t1, $zero, 100 #temp = 100
LOOP: lw $t3, 0($s0) #temp1 = A[i]
    add $v0, $v0, $t3 #sum += temp1
    addi $s0, $s0, 4 #addr of A[i+1]
    addi $t0, $t0, 1 #i = i+1
    bne $t1, $t0, LOOP #if i < 100
```

A. 7  
B. 402  
C. 502  
D. 407  
E. 507
Number of instructions being executed?

• For the following C code snippet and its corresponding MIPS translation on the right hand side, how many operations will the processor execute in total to complete all loop iterations?

A. 7  Static instructions — how many instructions in the compiled program
B. 402  
C. 502  Dynamic instructions — how many instructions in the executed program
D. 407  
E. 507  

```c
for(i = 0; i < 100; i++)
{
    sum+=A[i];
}
```

```mips
and $t0, $t0, $zero # let i = 0
addi $t1, $zero, 100 # temp = 100
LOOP: lw   $t3, 0($s0)     # temp1 = A[i]
    add  $v0, $v0, $t3   # sum += temp1
    addi $s0, $s0, 4     # addr of A[i+1]
    addi $t0, $t0, 1     # i = i+1
    bne  $t1, $t0, LOOP  # if i < 100
```

37
Your turn

- For the C code below:
  ```c
  if(n == 1)
      ret = 1;
  else n = n - 1;
  ```
  How many of the following is the correct translation in MIPS? Assume that $t0$ has $n$ and $v0$ has $ret$.

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

A. ```
   ```
   addi $t0, $t0, -1
   bne $t0, $zero, L1
   L1: addi $v0, $zero, 1
   ```
B. ```
   ```
   addi $t0, $t0, -1
   beq $t0, $zero, L1
   L1: addi $v0, $zero, 1
   ```
C. ```
   ```
   addi $t1, $t1, 1
   bne $t0, $t1, L1
   addi $v0, $zero, 1
   L1: addi $t0, $t0, -1
   ```
D. ```
   ```
   addi $t0, $t0, -1
   beq $t0, $zero, L1
   L1: addi $v0, $zero, 1
   ```
E. ```
   ```
   addi $t0, $t0, -1
   beq $t0, $zero, L1
   L1: addi $v0, $t0, 1
   ```
For the C code below:
```
if(n == 1)
    ret = 1;
else n = n - 1;
```
How many of the following is the correct translation in MIPS?
Assume that $t0$ has $n$ and $v0$ has $ret$.

A. 0
B. 1
C. 2
D. 3
E. 4
Your turn

• For the C code below:
  ```c
  if(n == 1)
      ret = 1;
  else n = n - 1;
  ```

  How many of the following is the correct translation in MIPS?
  Assume that $t0$ has $n$ and $v0$ has $ret$.

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

  - addi $t0$, $t0$, -1
    bne $t0$, $zero$, L1
    L1: addi $v0$, $zero$, 1

  - addi $t0$, $t0$, -1
    beq $t0$, $zero$, L1
    L1: addi $v0$, $zero$, 1

  - addi $t1$, $t1$, 1
    bne $t0$, $t1$, L1
    addi $v0$, $zero$, 1
    L1: addi $t0$, $t0$, -1

  - addi $t0$, $t0$, -1
    beq $t0$, $zero$, L1
    L1: addi $v0$, $t0$, 1
Supporting function calls
## Subset of MIPS instructions

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Usage</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>add</td>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>addi $s1,$s2, 20</td>
<td>$s1 = $s2 + 20</td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 − $s3</td>
</tr>
<tr>
<td>Logical</td>
<td>and</td>
<td>and $s1, $s2, $s3</td>
<td>$s1 = $s2 &amp; $s3</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>or $s1, $s2, $s3</td>
<td>$s1 = $s2</td>
</tr>
<tr>
<td></td>
<td>andi</td>
<td>andi $s1, $s2, 20</td>
<td>$s1 = $s2 &amp; 20</td>
</tr>
<tr>
<td></td>
<td>sll</td>
<td>sll $s1, $s2, 10</td>
<td>$s1 = $s2 * 2^10</td>
</tr>
<tr>
<td></td>
<td>srl</td>
<td>srl $s1, $s2, 10</td>
<td>$s1 = $s2 / 2^10</td>
</tr>
<tr>
<td>Data Transfer</td>
<td>lw</td>
<td>lw $s1, 4($s2)</td>
<td>$s1 = mem[$s2+4]</td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>lw $s1, 4($s2)</td>
<td>mem[$s2+4] = $s1</td>
</tr>
<tr>
<td>Branch</td>
<td>beq</td>
<td>beq $s1, $s2, 25</td>
<td>if($s1 == $s2), PC = PC + 100</td>
</tr>
<tr>
<td></td>
<td>bne</td>
<td>bne $s1, $s2, 25</td>
<td>if($s1 != $s2), PC = PC + 100</td>
</tr>
<tr>
<td>Jump</td>
<td>jal</td>
<td>jal 25</td>
<td>$ra = PC + 4, PC = 100</td>
</tr>
<tr>
<td></td>
<td>jr</td>
<td>jr $ra</td>
<td>PC = $ra</td>
</tr>
</tbody>
</table>

The only type of instructions can access memory.

We use them to support function calls!
int main(int argc, char **argv)
{
    n = atoi(argv[0]);
    bar = rand();
    printf("%d\n", foo(n));
    return 0;
}

int foo(n)
{
    int i, sum=0;
    for(i = 0; i < n; i++) {
        sum+=i;
    }
    return sum;
}
Announcements

• Resources
  • Ask questions — piazza
  • Reading quizzes, turning in assignments — Canvas
  • Slides, schedule, assignment questions — Check our website
  • Video archive — Prof. Usagi’s Youtube channel

• Reading quizzes
  • Another reading quiz due tomorrow before class — just 1.6, 1.8, 1.10
  • Adjustments on reading quiz schedule — adding one more reading quiz but also drop more now — planning to drop 3 lowest to accommodate the delay of enrollments/system issues
  • If Canvas team did not add you into the system before 9:30pm, send me an e-mail by 10pm. Prof. Usagi will try to help you.