Where is “Computer Architecture and Engineering”?  

* Coordination of many *levels of abstraction*

---

**Anatomy: 5 components of any Computer**

- Processor  
  - Control ("brain")  
  - Datapath ("brawn")  

- Memory  
  - (where programs, data live when running)

- Devices  
  - Keyboard, Mouse  
  - Disk (where programs, data live when not running)  
  - Display, Printer

---

**Computer Technology - Dramatic Change!**

- Processor  
  - 2X in speed every 1.5 years (since '85); 100X performance in last decade.

- Memory  
  - DRAM capacity: 2x / 2 years (since '96); 64x size improvement in last decade.

- Disk  
  - Capacity: 2X / 1 year (since '97)  
  - 250X size in last decade.

---

**Tech. Trends: Microprocessor Complexity**

2X transistors/Chip Every 1.5 to 2.0 years  
Called "Moore’s Law"
**Project Focus**
- Design Intensive Class ---
  100 to 150 hours per semester per student (250 before)
- MIPS Instruction Set ---> FPGA implementation

- Modern CAD System:
  Schematic capture and Simulation
  - Computer-based “breadboard”
  - Behavior over time
  - Before construction
  - Xilinx FPGA board
  - Running design at 10 to 25 MHz (~ state-of-the-art clock rate a decade ago)

**Project Simulates Industrial Environment**
- Project teams have 4 or 5 members in same discussion section
- Must work in groups as in “the real world”
- Communicate with colleagues (team members)
- Communication problems are natural
- What have you done?
- What answers you need from others?
- You must document your work!!!
- Everyone must keep an on-line notebook
- Communicate with supervisor (TAs)
  - How is the team’s plan?
  - Short progress reports are required:
    - What is the team’s game plan?
    - What is each member’s responsibility?

**CS152: So what’s in it for me?**
- Build a real computer!
- In-depth understanding of the inner-workings of computers & trade-offs at HW/SW boundary
  - Insight into fast/slow operations that are easy/hard to implement in hardware (HW)
- Experience with the **design process** in the context of a large complex (hardware) design.
  - Functional Spec --> Control & Datapath --> Physical implementation
  - Modern CAD tools
  - Make 32-bit RISC processor in actual hardware
- Learn to work as team, with manager (TA)
- Designer’s "Conceptual" toolbox.

**Conceptual tool box?**
- Evaluation Techniques
- Levels of translation (e.g., Compilation)
- Levels of Interpretation (e.g., Microprogramming)
- Hierarchy (e.g. registers, cache, mem, disk, tape)
- Pipelining and Parallelism
- Indirection and Address Translation
- Synchronous / Asynchronous Control Transfer
- Timing, Clocking, and Latching
- CAD Programs, Hardware Description Languages, Simulation
- Static / Dynamic Scheduling
- Physical Building Blocks (e.g., Carry Lookahead)
- Understanding Technology Trends / FPGAs

**Format: Lecture - Disc - Lecture - Lab**
- Mon Labs due
- Tue Lecture
- Wed Homeworks due
- Thu Lecture
- Fri Discussion Section/Lab demo
- There IS discussion this week…;
- **Prerequisite Quiz in 10 days** (Friday in discussion section)

**2 Discussion Sections**
1. Noon - 2 PM 85 Evans (Brandon)
2. 2 PM - 4 PM 87 Evans (Doug)
2-hour discussion section for later in term. Early sections may end in 1 hour. Make sure that you are free for both hours however!
- **Project team must be in same section!**
To Do Now: Fill out Survey with Photo

°Survey is up now on the website in the "News" section at the top

°The deadline for turning in survey is Tuesday 9/7
  • Photo + survey of interesting items
  • Survey of your views on cheating to help with departmental discussion

Typical 80-minute Lecture Format

°18-Minute Lecture + 2-Min admin break
°20-Minute Lecture + 10-Min Peer instruct.
°25-Minute Lecture + 5-Min wrap-up
°We’ll come to class early & try to stay after to answer questions

<table>
<thead>
<tr>
<th>Attention</th>
<th>20 min.</th>
<th>Break</th>
<th>Time</th>
<th>Next thing</th>
<th>“In conclusion”</th>
</tr>
</thead>
</table>

Tried-and-True Technique: Peer Instruction

°Increase real-time learning in lecture, test understanding of concepts vs. details
°As complete a “segment” ask multiple choice question
  • 1-2 minutes to decide yourself
  • 3-4 minutes in pairs/triples to reach consensus. Teach each other!
  • 2-3 minute discussion of answers, questions, clarifications

Homeworks and Labs/Projects

°Homework exercises (every 2 weeks)
°Lab Projects (every ~2 weeks)
  • Lab 1 Write diagnostics to debug bad SPIM
  • Lab 2 Single Cycle Processor
  • Lab 3 Pipelined Processor
  • Lab 4 Cache and Memory Interface
°All exercises, reading, homeworks, projects on course web page

Project/Lab Summary

°Tool Flow runs on PCs in 119 and 125 Cory, but 119 Cory is primary CS152 lab
°Get instructional UNIX/PC account now ("name account"); get in discussion
°End of semester Project finale:
  • Demo
  • Oral Presentation
  • Head-to-head Race
  • Final Report

Course Exams

°Reduce the pressure of taking exams
  • Midterms: Tue October 12th and Tue Nov. 23rd in 306 Soda
  • 3 hrs to take 1.5-hr test (5:30-8:30 PM)
  • Our goal: test knowledge vs. speed writing
  • Review meetings: Sunday before?
  • Both mid-terms can bring summary sheets
°Students/Staff meet over pizza after exam at LaVals!
  • Allow us to meet you
  • We’ll buy!
Grading

- Grade breakdown
  - Two Midterm Exams: 32% (combined)
  - Labs: 30%
  - Final Project: 20%
  - Homeworks: 8%
  - Group/Class Participation: 10%
- No late homeworks or labs: our goal grade, return in 1 week
- Grades posted on home page/glookup?
  - Written/email request for changes to grades
- EECS GPA guideline upper div. class: 2.7 to 3.1
  - average 152 grade = B/B+; set expectations accordingly

Our Goals

- Show you how to understand modern computer architecture in its rapidly changing form
- Show you how to design by leading you through the process on challenging design problems and by examining real designs
- Learn how to test and to design for test
- Reduce workload from prior semesters yet more computers working for head-to-head race
  - Simpler homeworks
  - 4 labs vs. 6 labs
  - Simpler final project target

Course Problems...Cheating

- What is cheating?
  - Studying together in groups is encouraged
  - Work must be your own (or your group’s own)
  - Common examples of cheating: work together on wording of answer to homework, running out of time on a assignment and then pick up output, take homework from box and copy, person asks to borrow solution “just to take a look”, copying an exam question, copy old projects, ...
- Homeworks/labs/projects/exams # points varies: 0 and possibly F in course
- Inform Chair and Office of Student Conduct

EECS Policy: [www.eecs.berkeley.edu/Policies/acad.dis.shtml](http://www.eecs.berkeley.edu/Policies/acad.dis.shtml)

Copying all or part of another person’s work, or using reference material not specifically allowed, are forms of cheating and will not be tolerated. A student involved in an incident of cheating will be notified by the instructor and the following policy will apply:

1. The Instructor may take actions such as:
   A. require repetition of the subject work,
   B. assign an F grade or a ‘zero’ grade to the subject work,
   C. for serious offenses, assign an F grade for the course.
2. The recommended action for cheating on examinations or term papers is 1(C).
3. The Instructor must inform the student and the Department Chair in writing of the incident, the action taken, if any, and the student’s right to appeal to the Chair of the Department Grievance Committee or to the Director of the Office of Student Conduct.
4. The Instructor retains copies of any written evidence or observation notes.
5. The Department Chair must inform the Director of the Office of Student Conduct of the incident, the student’s name, & action taken by the instructor.
6. The Office of Student Conduct may choose to conduct a formal hearing on the incident and to assess a penalty for misconduct.
7. The Department will recommend that students involved in a second incident of cheating be dismissed from the University.

Text

- Required: *Computer Organization and Design: The Hardware/Software Interface, 3rd Edition*, Patterson and Hennessy (COD)
  - 3rd edition $20 less than 2nd edition ($56 discounted vs. $100+ for competition)
  - CD inside book includes manuals, appendices, simulators, CAD, ...
  - “Green card” summarizes MIPS
  - Readings on web page inst.eecs.berkeley.edu/~cs152
  • Need 3rd edition? Yes, since changed almost every page, CD, verilog, ...

MIPS I

Instruction set
MIPS I: Operation Overview

**Arithmetic Logical:**
- Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU
- AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, LUI
- SLL, SRL, SRA, SLLV, SRLV, SRAV

**Memory Access:**
- LB, LBU, LH, LHU, LW, LWL, LWR
- SB, SH, SW, SWL, SWR

MIPS logical instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>$1 &amp; $2 &amp; $3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 reg. operands; Logical AND</td>
</tr>
<tr>
<td>or</td>
<td>$1 | $2 | $3</td>
<td>$1 = $2 | $3</td>
<td>3 reg. operands; Logical OR</td>
</tr>
<tr>
<td>xor</td>
<td>$1 \oplus $2 \oplus $3</td>
<td>$1 = \neg ($2 | $3)</td>
<td>3 reg. operands; Logical XOR</td>
</tr>
<tr>
<td>and immediate</td>
<td>!1 &amp; !2 $1 = !2 &amp; !3</td>
<td>Logical AND reg, constant</td>
<td></td>
</tr>
<tr>
<td>or immediate</td>
<td>!1 | !2 $1 = !2</td>
<td>Logical OR reg, constant</td>
<td></td>
</tr>
<tr>
<td>xor immediate</td>
<td>!1 \oplus !2 $1 = !2 \oplus !3</td>
<td>Logical XOR reg, constant</td>
<td></td>
</tr>
<tr>
<td>shift left logical</td>
<td>!1 &lt;&lt; !2</td>
<td>Shift left by constant</td>
<td></td>
</tr>
<tr>
<td>shift right logical</td>
<td>!1 &gt;&gt; !2</td>
<td>Shift right by constant</td>
<td></td>
</tr>
<tr>
<td>shift right arithm.</td>
<td>!1 &gt;&gt; !2</td>
<td>Shift right (sign extend)</td>
<td></td>
</tr>
<tr>
<td>shift left logical</td>
<td>!1 &lt;&lt; !2</td>
<td>Shift left by variable</td>
<td></td>
</tr>
<tr>
<td>shift right logical</td>
<td>!1 &gt;&gt; !2</td>
<td>Shift right by variable</td>
<td></td>
</tr>
<tr>
<td>shift right arithm.</td>
<td>!1 &gt;&gt; !2</td>
<td>Shift right arith. by variable</td>
<td></td>
</tr>
</tbody>
</table>

Q: Can some multiply by 2^i? Divide by 2^i? Invert?

MIPS Reference Data: CORE INSTRUCTION SET (1)

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNE-MON-IC</th>
<th>FOR-MAT</th>
<th>OPERATION (in Verilog)</th>
<th>OPCOD/E/FUNC T (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>add</td>
<td>R</td>
<td>R[rd] = R[rs] + R[rt] (1)</td>
<td>0 / 20hex</td>
</tr>
<tr>
<td>Add Immediate</td>
<td>addi</td>
<td>I</td>
<td>R[rt] = R[rs] + SignExtImm (1)(2)</td>
<td>8hex</td>
</tr>
<tr>
<td>Branch On Equal</td>
<td>beq</td>
<td>I</td>
<td>if[R[rs] == R[rt]] PC=PC+4+BranchAddr (4)</td>
<td>8hex</td>
</tr>
</tbody>
</table>

(1) May cause overflow exception
(2) SignExtImm = [16{immediate[15]}], immediate
(3) ZeroExtImm = [16{1b'0}], immediate
(4) BranchAddr = [14{immediate[15]}], immediate, 2'b0

MIPS data transfer instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw $1, 300($4), $3</td>
<td>Store word</td>
<td></td>
</tr>
<tr>
<td>sh $2, 502($3), $3</td>
<td>Store half</td>
<td></td>
</tr>
<tr>
<td>sb $1, 41($3), $2</td>
<td>Store byte</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $1, 30($2)</td>
<td>Load word</td>
<td></td>
</tr>
<tr>
<td>lh $1, 40($3)</td>
<td>Load halfword</td>
<td></td>
</tr>
<tr>
<td>lhu $1, 40($3)</td>
<td>Load halfword unsigned</td>
<td></td>
</tr>
<tr>
<td>lb $1, 40($3)</td>
<td>Load byte</td>
<td></td>
</tr>
<tr>
<td>lbu $1, 40($3)</td>
<td>Load byte unsigned</td>
<td></td>
</tr>
</tbody>
</table>

lui $1, 40 | Load Upper Immediate (16 bits shifted left by 16)

Q: Why need lui?

MIPS arithmetic instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$1 &amp; $2, $3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 operands; exception possible</td>
</tr>
<tr>
<td>subtract</td>
<td>$1 &amp; $2, $3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 operands; exception possible</td>
</tr>
<tr>
<td>add immediate</td>
<td>$1, $2, $3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 operands; exception possible</td>
</tr>
<tr>
<td>add unsigned</td>
<td>$1, $2, $3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 operands; no exception</td>
</tr>
<tr>
<td>subtract unsigned</td>
<td>$1, $2, $3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 operands; no exception</td>
</tr>
<tr>
<td>multiply</td>
<td>$1 \times $2</td>
<td>$1 \times $2</td>
<td>64-bit signed product</td>
</tr>
<tr>
<td>multiply unsigned</td>
<td>$1 \times $2</td>
<td>$1 \times $2</td>
<td>64-bit unsigned product</td>
</tr>
<tr>
<td>divide</td>
<td>$1 / $2</td>
<td>$1 / $2</td>
<td>$1 / $2 = quotient, $1 / $2 = remainder</td>
</tr>
<tr>
<td>divide unsigned</td>
<td>$1 / $2</td>
<td>$1 / $2</td>
<td>$1 / $2 = quotient, $1 / $2 = remainder</td>
</tr>
<tr>
<td>Move from Hi</td>
<td>$1 &amp; Lo</td>
<td>$1 &amp; Lo</td>
<td>Used to get copy of Hi</td>
</tr>
<tr>
<td>Move from Lo</td>
<td>$1 &amp; Hi</td>
<td>$1 &amp; Lo</td>
<td>Used to get copy of Lo</td>
</tr>
</tbody>
</table>

Q: Which add for address arithmetic? Which add for integers?
When does MIPS sign extend?

° When value is sign extended, copy upper bit to full value:
Examples of sign extending 8 bits to 16 bits:

00000010 00000000 00001010
00000011 00000000 00001010

° When is an immediate operand sign extended?
• Arithmetic instructions (add, sub, etc.) always sign extend immediates even for the
  unsigned versions of the instructions.
• Logical instructions do not sign extend immediates (They are zero extended)
• Load/Store address computations always sign extend immediates

° Multiply/Divide have no immediate operands however:
• "unsigned" treat operands as unsigned

° The data loaded by the instructions lb and lh are extended as follows
  ("unsigned" don't extend):
• lbu, lhu are zero extended
• lb, lh are sign extended

Q: Then what is does add unsigned (addu) mean since not immediate?

MIPS Compare and Branch

° Compare and Branch
• BEQ rs, rt, offset if R[rs] == R[rt] then PC-relative branch
• BNE rs, rt, offset !=

° Compare to zero and Branch
• BLEZ rs, offset if R[rs] <= 0 then PC-relative branch
• BGTZ rs, offset >
• BGEZAL rs, offset if R[rs] < 0 then branch and link (into R 31)

° Remaining set of compare and branch ops take two instructions
° Almost all comparisons are against zero!

Signed vs. Unsigned Comparison

$4 = 0…00 0000 0000 0000 0001$
two
$5 = 0…00 0000 0000 0000 0010$
two
$6 = 1…11 1111 1111 1111 1111$
two

° After executing these instructions:
slt $4, $2, $1 ; if ($2 < $1) $4=1; else $4=0
slt $5, $3, $1 ; if ($3 < $1) $5=1; else $5=0
sltu $6, $2, $1 ; if ($2 < $1) $6=1; else $6=0
sltu $7, $3, $1 ; if ($3 < $1) $7=1; else $7=0

° What are values of registers $4 - $7? Why?
$4 = 0 ; 5 = 0 ; 6 = 0 ; 7 = 0$

MIPS jump, branch, compare instructions

branch on equal beq $1,$2,100
if ($1 == $2) go to PC+4+100
Equal test; PC relative branch

branch on not eq. bne $1,$2,100
if ($1!= $2) go to PC+4+100
Not equal test; PC Relative

set on less than slt $1,$2,$3
if ($2 < $3) $1=1; else $1=0
Compare less than; 2's comp.

set on less than imm. slti $1,$2,100
if ($2 < 100) $1=1; else $1=0
Compare < constant; 2's comp.

set on less than uns. sltu $1,$2,$3
if ($2 < $3) $1=1; else $1=0
Compare < constant; Natural numbers

Jump j 10000
go to 10000
Jump to target address

Jump register jr $31
go to $31
For switch, procedure return

Jump and link jal 10000 $31 = PC + 4; go to 10000
For procedure call

Signed vs. Unsigned Comparison

$4 = 0…00 0000 0000 0000 0001$
two
$5 = 0…00 0000 0000 0000 0010$
two
$6 = 1…11 1111 1111 1111 1111$
two

° After executing these instructions:
slt $4, $2, $1 ; if ($2 < $1) $4=1; else $4=0
slt $5, $3, $1 ; if ($3 < $1) $5=1; else $5=0
sltu $6, $2, $1 ; if ($2 < $1) $6=1; else $6=0
sltu $7, $3, $1 ; if ($3 < $1) $7=1; else $7=0

° What are values of registers $4 - $7? Why?
$4 = 0 ; 5 = 1 ; 6 = 0 ; 7 = 0$
MIPS assembler register convention

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Usage</th>
<th>Preserved across a call?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>0</td>
<td>the value 0</td>
<td>n/a</td>
</tr>
<tr>
<td>$v0-$v1</td>
<td>2-3</td>
<td>return values</td>
<td>no</td>
</tr>
<tr>
<td>$s0-$s3</td>
<td>4-7</td>
<td>arguments</td>
<td>no</td>
</tr>
<tr>
<td>$t0-$t7</td>
<td>8-15</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$s0-$s7</td>
<td>16-23</td>
<td>saved</td>
<td>yes</td>
</tr>
<tr>
<td>$t18-$t19</td>
<td>24-25</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>stack pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>return address</td>
<td>yes</td>
</tr>
</tbody>
</table>

° "caller saved"
° "callee saved"

On Green Card in Column #2 at bottom

Peer Instruction: $s3=i, $s4=j, $s5=@A

Loop: addiu $s4, $s4, 1     # j = j + 1
      add $t1, $t1, $s5      # $t1 = A[i]
      lw $s10, 8($t1)        # $s10 = A[i] + 1
      beq $s10, $0, Loop    # goto Loop [not delayed]
      addiu $s3, $s3, 1     # i = i + 1
      slti $t1, $s10, 9     # $t1 = A[i] < 10
      beq $t1, $0, Loop     # goto Loop [not delayed]

What C code properly fills in the blank in loop on right?

1: A[i++] >= 10
2: A[i_] >= 10 || A[i++] < 0
3: A[i_] >= 10 || A[i_] < 0
4: A[i++] >= 10 || A[i_] < 0
5: A[i_] >= 10 & & A[i_] < 0
6: None of the above

Instruction Formats

° I-format: used for instructions with immediates, lw and sw (since the offset counts as an immediate), and the branches (beq and bne),
  - (but not the shift instructions; later)
° J-format: used for j and jal
° R-format: used for all other instructions
  - It will soon become clear why the instructions have been partitioned in this way.

R-Format Instructions (1/2)

° Define “fields” of the following number of bits each: 6 + 5 + 5 + 5 + 6 = 32

|       | 6 | 5 | 5 | 5 | 6 |
° For simplicity, each field has a name:

| opcode | rs | rt | rd | shamt | funct |

R-Format Instructions (2/2)

° More fields:
  - rs (Source Register): generally used to specify register containing first operand
  - rt (Target Register): generally used to specify register containing second operand (note that name is misleading)
  - rd (Destination Register): generally used to specify register which will receive result of computation

Peer Instruction: $s3=i, $s4=j, $s5=@A

Loop: addiu $s4, $s4, 1     # j = j + 1
      add $t1, $t1, $s5      # $t1 = A[i]
      lw $s10, 8($t1)        # $s10 = A[i] + 1
      beq $s10, $0, Loop    # goto Loop [not delayed]
      addiu $s3, $s3, 1     # i = i + 1
      slti $t1, $s10, 9     # $t1 = A[i] < 10
      beq $t1, $0, Loop     # goto Loop [not delayed]

What C code properly fills in the blank in loop on right?

1: A[i++] >= 10
2: A[i_] >= 10 || A[i++] < 0
3: A[i_] >= 10 || A[i_] < 0
4: A[i++] >= 10 || A[i_] < 0
5: A[i_] >= 10 & & A[i_] < 0
6: None of the above

Instruction Formats

° I-format: used for instructions with immediates, lw and sw (since the offset counts as an immediate), and the branches (beq and bne),
  - (but not the shift instructions; later)
° J-format: used for j and jal
° R-format: used for all other instructions
  - It will soon become clear why the instructions have been partitioned in this way.

R-Format Instructions (1/2)

° Define “fields” of the following number of bits each: 6 + 5 + 5 + 5 + 6 = 32

|       | 6 | 5 | 5 | 5 | 6 |
° For simplicity, each field has a name:

| opcode | rs | rt | rd | shamt | funct |

R-Format Instructions (2/2)

° More fields:
  - rs (Source Register): generally used to specify register containing first operand
  - rt (Target Register): generally used to specify register containing second operand (note that name is misleading)
  - rd (Destination Register): generally used to specify register which will receive result of computation
J-Format Instructions (1/2)

- Define “fields” of the following number of bits each:
  - 6 bits
  - 26 bits

- As usual, each field has a name:
  - opcode
  - target address

- Key Concepts
  - Keep opcode field identical to R-format and I-format for consistency.
  - Combine all other fields to make room for large target address.

J-Format Instructions (2/2)

- Summary:
  - New PC = (PC[31..28], target address, 00)

- Understand where each part came from!

- Note: In Verilog,
  \{, , \} means concatenation
  \{ 4 bits , 26 bits , 2 bits \} = 32 bit address

  \{( 1010, 11111111111111111111111111, 00 \}
  = 1010111111111111111111111111111100

  We use Verilog in this class

R-Format Example

- MIPS Instruction:
  \text{add \$8, \$9, \$10}

Decimal number per field representation:

<table>
<thead>
<tr>
<th>0</th>
<th>9</th>
<th>10</th>
<th>8</th>
<th>0</th>
<th>32</th>
</tr>
</thead>
</table>

Binary number per field representation:

<table>
<thead>
<tr>
<th></th>
<th>0000 00</th>
<th>0100 10</th>
<th>0110 01</th>
<th>0100 00</th>
<th>0000 00</th>
<th>1000 00</th>
</tr>
</thead>
</table>

hex representation: \(012A \text{ } 4020\) hex
decimal representation: \(19,546,144\)

On Green Card: Format in column 1, opcodes in column 3

Green Card: OPCODES, BASE CONVERSION, ASCII (3)

<table>
<thead>
<tr>
<th>MIPS opcode (31:26)</th>
<th>(1) MIPS funct (5:0)</th>
<th>(2) MIPS funct (5:0)</th>
<th>Binary (31:26)</th>
<th>Deci- mal (31:26)</th>
<th>Hexa- decimal (31:26)</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sll</td>
<td>add</td>
<td>00 0000</td>
<td>0 0</td>
<td>NUL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set</td>
<td>mul</td>
<td>00 0010</td>
<td>2 2</td>
<td>STX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sync</td>
<td>floor.w</td>
<td>00 1111</td>
<td>15</td>
<td>f</td>
<td>SI</td>
</tr>
<tr>
<td></td>
<td>lbu</td>
<td>cvt.w</td>
<td>10 0100</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
</tbody>
</table>

(1) opcode(31:26) == 0
(2) opcode(31:26) == 17 ten (11 hex);
  if fmt(25:21)==16 ten (10 hex) \(f = s\) (single);
  if fmt(25:21)==17 ten (11 hex) \(f = d\) (double)

Note: 3-in-1 - Opcodes, base conversion, ASCII!

Green Card

- Green card /n./ [after the “IBM System/360 Reference Data” card]
  A summary of an assembly language, even if the color is not green. For example,

  “I’ll go get my green card so I can check the addressing mode for that instruction.”

www.jargon.net

Image from Dave’s Green Card Collection:
http://www.planetmvs.com/greencard/

Peer Instruction

Which instruction has same representation as \(35\) \(_{10}\) ?

A. \text{add \$0, \$0, \$0}
B. \text{subu \$0, \$0, \$0}
C. \text{lwr \$0, 0(\$0)}
D. \text{addi \$0, \$0, 35}
E. \text{subu \$0, \$0, \$0}
F. Trick question!

Instructions are not numbers

Use Green Card handout to answer
Peer Instruction

Which instruction has same representation as 35?

A. add $0, $0, $0
B. subu $s0,$s0,$s0
C. lw $0, 0($0)
D. addi $0, $0, 35
E. subu $0, $0, $0

Trick question! Instructions are not numbers

Registers numbers and names:
0: $0, 8: $t0, 9:$t1, ..,16: $s0, 17: $s1, .. , 23: $s7

Opcodes and function fields (if necessary)

add: opcode = 0, funct = 32
subu: opcode = 0, funct = 35
addi: opcode = 8
lw: opcode = 35

Peer Instruction

Which instruction bit pattern = number 35?

A. add $0, $0, $0
B. subu $s0,$s0,$s0
C. lw $0, 0($0)
D. addi $0, $0, 35
E. subu $0, $0, $0

Trick question! Instructions are not numbers

Registers numbers and names:
0: $0, 8: $t0, 9:$t1, ..,16: $s0, 17: $s1, .. , 23: $s7

Opcodes and function fields

add: opcode = 0, function field = 32
subu: opcode = 0, function field = 35
addi: opcode = 8
lw: opcode = 35

Branch & Pipelines

li $3, #7
sub $4, $4, 1
be $4, LL
addi $5, $3, 1
ll: slt $1, $3, $5

By the end of Branch instruction, the CPU knows whether or not the branch will take place. However, it will have fetched the next instruction by then, regardless of whether or not a branch will be taken. Why not execute it?

Delayed Branches

li $3, #7
sub $4, $4, 1
be $4, LL
addi $5, $3, 1
ll: slt $1, $3, $5

"In the “Raw” MIPS, the instruction after the branch is executed even when the branch is taken
- This is hidden by the assembler for the MIPS "virtual machine"
- allows the compiler to better utilize the instruction pipeline (??)

Jump and link (jal inst):
- Put the return addr. Into link register ($31):
  - PC+4 (logical architecture)
  - PC+8 physical ("Raw") architecture ⇒ delay slot executed
- Then jump to destination address

Filling Delayed Branches

Branch: Inst Fetch D. Add Op Fetch Execute
execute successor or continue

Single delay slot impacts the critical path

- Compiler can fill a single delay slot with a useful instruction 50% of the time.
- try to move down from above
- move up from target, if safe

Is this violating the ISA abstraction?

Summary: Salient features of MIPS I

- 32-bit fixed format inst (3 formats)
- 32 32-bit GPR (R0 contains zero) and 32 FP registers (and HI LO)
- 3-address, reg-reg arithmetic instr.
- Single address mode for load/store: base + displacement
- no indirection, scaled
- 16-bit immediate plus LUI
- Simple branch conditions
- compare against zero or two registers for =, >
- no integer condition codes
- Delayed branch
- execute instruction after a branch (or jump) even if the branch is taken
  (Compiler can fill a delayed branch with useful work about 50% of the time)
And in conclusion...

- Continued rapid improvement in Computing
  - 2X every 1.5 years in processor speed;
    every 2.0 years in memory size;
    every 1.0 year in disk capacity;
  - Moore's Law enables processor, memory
    (2X transistors/chip/ ~1.5 to 2.0 yrs)

- 5 classic components of all computers
  Control Datapath Memory Input Output

Processor