Class 5: 
Simple Risc Computer (SRC)
Register Transfer Notation

Topics
- Addressing Modes
- Example machine description - SRC
- Formal description languages – RTN
- Lab conflict revisited

Addressing Modes
- Access paths to operands
  - Memory
  - CPU registers
- Remember
  - To access memory
    - CPU generates address (effective address)
    - Sends to memory subsystem
  - HLL programs structure data in different ways

Immediate Addressing
- Instruction contains operand
  - load #3

Direct Addressing
- Instruction contains address
  - load A

Indirect Addressing
- Instruction contains address of the address of operand
  - load (A)
Register Direct Addressing
- Register contains operand
- Load R2

Register Indirect Addressing
- Register contains address of operand
- Load R1

Displacement Addressing
- Aka based addressing
- Operand address = register + offset
- Load 4[R1]

Relative Addressing
- Operand address = register + offset
- Load rel 4[PC]

Simple Risc Computer
Machine Description

SRC
- 32 general purpose registers (32-bit)
- PC
- IR
- 32-bit words (4 bytes) can be fetched or stored
SRC Instruction Categories

- Load and store
  - ld, ldr, la, lar
  - st, str
- Branch
  - Unconditional or conditional (= 0, <> 0, >= 0, or < 0)
  - br, brl
- Arithmetic
  - add, sub, neg, addi
- Logical and shift
  - and, ori, or, not, shr, sha, shl, shc
- Miscellaneous – nop, stop

Instruction Details

- Instruction == 32 bits
  - Opcode – 5 bits
    - 32 possible operations, only 23 defined
- Register fields – 5 bits (32 registers = 2^5)
  - Some instructions have unused bits
- Notation
  - R[x] – value stored in register x
  - M[x] – value stored at word x in memory

Load and Store

<table>
<thead>
<tr>
<th>Instruction</th>
<th>op</th>
<th>ra</th>
<th>rb</th>
<th>cl</th>
<th>Meaning</th>
<th>Addressing Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld r, la</td>
<td>31</td>
<td>27</td>
<td>22</td>
<td>16</td>
<td>R[la] = M[rb]</td>
<td>Direct</td>
</tr>
<tr>
<td>ld r, la</td>
<td>31</td>
<td>27</td>
<td>22</td>
<td>16</td>
<td>R[la] = M[rb]</td>
<td>Direct</td>
</tr>
<tr>
<td>st r, la</td>
<td>31</td>
<td>27</td>
<td>22</td>
<td>16</td>
<td>R[la] = M[rb]</td>
<td>Direct</td>
</tr>
<tr>
<td>st r, la</td>
<td>31</td>
<td>27</td>
<td>22</td>
<td>16</td>
<td>R[la] = M[rb]</td>
<td>Direct</td>
</tr>
</tbody>
</table>

Miscellaneous Insts

- nop – 0
- stop - 31

Arithmetic and Logic

- 1-operand
  - neg ra, rc
    - Negate: 2's complement
  - not ra, rc
    - Not: 1's complement
- 2-operand
  - add ra, rb, rc
    - R[ra] = R[rb] + R[rc]
  - sub ra, rb, rc
    - 2's complement subtract
  - and ra, rb, rc
    - Logical AND
  - or ra, rb, rc
    - Logical OR

2's Complement Review

<table>
<thead>
<tr>
<th>Sign</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11111111</td>
</tr>
<tr>
<td>0</td>
<td>00000001</td>
</tr>
<tr>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>1</td>
<td>11111111</td>
</tr>
<tr>
<td>1</td>
<td>11111110</td>
</tr>
<tr>
<td>1</td>
<td>00000011</td>
</tr>
<tr>
<td>1</td>
<td>00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sign</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>-127</td>
</tr>
<tr>
<td>1</td>
<td>-128</td>
</tr>
</tbody>
</table>
Arithmetic and Logic (cont’d)

• 2-operand (cont’d)
  addi ra, rb, c2 ; add Immediate
  andi ra, rb, c2 ; logical and, immediate
  ori ra, rb, c2 ; logical or, immediate

Shift Instructions

shr ra, rb, rc ; shifts 0s from left
shra ra, rb, rc ; arithmetic: shifts copy
  of msb
shr ra, rb, count
shra ra, rb, count

Also shl, shc

Branch Instructions

br rb, rc, c3 ; branch to R[rb] if R[rc] meets c3 is met

Condition

0 Never
1 Always
2 if R[rc] == 0
3 if R[rc] != 0
4 if R[rc] >= 0
5 if R[rc] < 0

Unconditional Branch Example

C: goto Label3

SRC:
lar r0, Label3 ; put branch target address into tgt reg.
br r0 ; and branch

Label3 • • •

Conditional Branch Example

#define Cost 125  
pseudo ops:
if(X<0) X = -X
.equ .equ
.org .org
.dw .org
Cost: .equ 125 ; Cost = 125
.dw 1000 ; Next word location 1000
X: .dw 1 ; reserve space for X
.org 5000 ; load prog at 5000
lar r0, Over ; load address of Over into r0
ld r1, X ; load X into r1
brpl r0, r1 ; if r1 >= 0, branch to Over
neg r1, r1 ; r1 = -r1 (X = -X)
Over:

SRC Opcode Summary
### SRC Opcode Summary (2)

<table>
<thead>
<tr>
<th>Op code</th>
<th>Op code</th>
<th>Op code</th>
<th>Op code</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>and</td>
<td>21</td>
<td>shr</td>
</tr>
<tr>
<td>18</td>
<td>or</td>
<td>22</td>
<td>not</td>
</tr>
<tr>
<td>19</td>
<td>ori</td>
<td>23</td>
<td>shr</td>
</tr>
<tr>
<td>20</td>
<td>not</td>
<td>24</td>
<td>shl</td>
</tr>
<tr>
<td>25</td>
<td>shc</td>
<td>26</td>
<td>stop</td>
</tr>
<tr>
<td>27</td>
<td>shra</td>
<td>28</td>
<td>nop</td>
</tr>
</tbody>
</table>

### Register Transfer Languages

**Example:**

Register Transfer Notation

### Formal Description

- **Machine**
  - Data in registers and memory cells
  - moved around
  - transformed
- **Register transfer language**
  - Formally describes this process
  - Example: RTN

### Processor State

- **PC<31..0>:**
- **IR<31..0>:** // flip-flop
- **Run:** // external signal
- **R[0..31]<31..0>:**

### Instruction Formats

- **op<4..0> := IR<31..27>:**
- **ra<4..0> := IR<26..22>:**
- **rb<4..0> := IR<21..17>:**
- **rc<4..0> := IR<16..12>:**
- **c1<21..0> := IR<21..0>:**
- **c2<16..0> := IR<16..0>:**
- **c3<11..0> := <11..0>:**

### Main Memory

- **Mem[0..2^{32}-1]<7..0>:**
- **M[x]<31..0> := Mem[x]#Mem[x+1]#Mem[x+2]#Mem[x+3]:**

### Instruction Formats
Effective Address Calculation

- Displacement
disp<31..0> := ((rb=0) → c2<16..0> {sign extend}:
  (rb!=0) → R[rb]+c2<16..0> {sign extend, 2's complement}):

- PC Relative
rel<31..0> := PC<31..0> + c1<21..0> {sign extend, 2's comp}:

Instruction Interpretation

Fetch/Execute Cycle

instruction_interpretation := (
¬Run ^ Strt → Run ← 1; instruction_interpretation):
Run → (IR ← M[PC]; PC ← PC+ 4;
instruction_execution)) ;

Instruction Execution

instruction_execution := (
ld(:, op=1) → R[ra] ← M[disp]:
...
cond := (c3<2..0>=0 → 0:
  ...):
br(:, op=9) → (cond → PC ← R[rb]):
  ...:
stop(:, op=31) → Run ← 0
); instruction_interpretation:

Summary

- Processor and memory state
- Formats and interpretation of data in registers
- Instruction interpretation sequence
- Description of instructions

SRC vs. RTN – What’s the diff?

- SRC – machine language
  - Can be run on hardware or simulator
- RTN – metalanguage
  - Language which specifies a language

Next Class

- Register Transfers and Logic Circuits