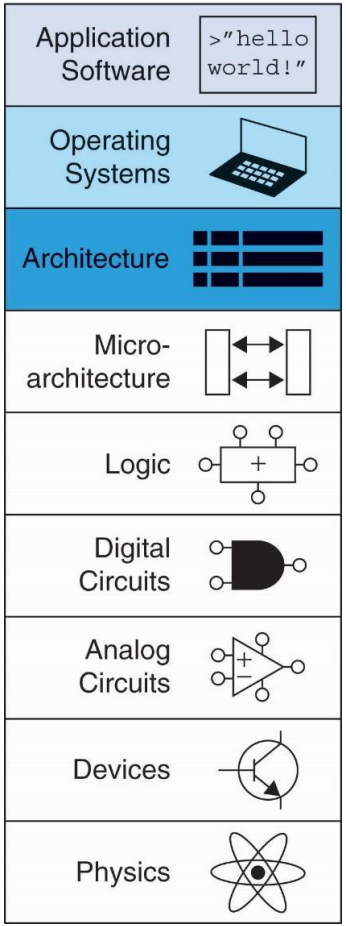
COSC175 (Systems I): Computer Organization & Design

Professor Lillian Pentecost
Fall 2024

Warm-Up October 29

- Where we were
  - Writing RISC-V assembly programs to make direct use of our HW architecture
- Where we are going
  - Writing MORE RISC-V assembly programs to make direct use of our HW architecture
- Logistics, Reminders
  - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
  - LP Office hours M 9-10:30AM, Th 2:30-4PM
  - Weekly Exercises (including programming exercises) will be posted Friday, due next week
  - Lab 5 - First Stage as Pre-Lab for next week **individually**
  - Lab 5 Report due November 4 10PM



Programming

- **High-level languages:**
  - e.g., C, Java, Python
  - Written at higher level of abstraction
- **High-level constructs:** loops, conditional statements, arrays, function calls
- **First, introduce instructions that support these:**
  - Logical operations
  - Shift instructions
  - Multiplication & division
  - Branches & Jumps

Branching

- Execute instructions out of sequence
- Types of branches:
  - **Conditional**
    - branch if equal (beq)
    - branch if not equal (bne)
    - branch if less than (blt)
    - branch if greater than or equal (bge)
  - **Unconditional**
    - jump (j)
    - jump register (jr)
    - jump and link (jal)
    - jump and link register (jalr)

We'll talk about these when discuss function calls

Conditional Branching

```
# RISC-V assembly
 addi s0, zero, 4     # s0 = 0 + 4 = 4
 addi s1, zero, 1     # s1 = 0 + 1 = 1
 slli s1, s1, 2    # s1 = 1 << 2 = 4
 beq  s0, s1, target # branch is taken
 addi s1, s1, 1          # not executed
 sub  s1, s1, s0      # not executed

target:          # label
 add  s1, s1, s0   # s1 = 4 + 4 = 8
```

Labels indicate instruction location. They can't be reserved words and must be followed by a colon (:)

The Branch Not Taken (bne)

```
# RISC-V assembly
   addi  s0, zero, 4          # s0 = 0 + 4 = 4
   addi  s1, zero, 1          # s1 = 0 + 1 = 1
   slli  s1, s1, 2            # s1 = 1 << 2 = 4
   bne   s0, s1, target    # branch not taken
   addi  s1, s1, 1            # s1 = 4 + 1 = 5
   sub   s1, s1, s0     # s1 = 5 - 4 = 1

target:
   add   s1, s1, s0     # s1 = 1 + 4 = 5
```

Unconditional Branching (`j`)

```
# RISC-V assembly
   j        target          # jump to target
   srai     s1, s1, 2          # not executed
   addi     s1, s1, 1          # not executed
   sub      s1, s1, s0      # not executed

target:
   add   s1, s1, s0   # s1 = 1 + 4 = 5
```

Conditional Statements & Loops

- **Conditional Statements**
  - if statements
  - if/else statements
- **Loops**
  - while loops
  - for loops

If Statement

C Code

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


f = f - i;
```

RISC-V assembly code

```
# s0 = f, s1 = g, s2 = h
# s3 = i, s4 = j
    bne s3, s4, L1
    add s0, s1, s2

L1:
    sub s0, s0, s3
```

Assembly tests opposite case (`i != j`) of high-level code (`i == j`)

If/Else Statement

C Code

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

else
  f = f - i;
```

RISC-V assembly code

```
# s0 = f, s1 = g, s2 = h
# s3 = i, s4 = j
    bne  s3, s4, L1
    add  s0, s1, s2
    j    done

L1:
    sub  s0, s0, s3
done:
```

Assembly tests opposite case (`i != j`) of high-level code (`i == j`)

While Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

RISC-V assembly code

```
# s0 = pow, s1 = x

        addi s0, zero, 1
        add  s1, zero, zero
        addi t0, zero, 128
while:
        beq  s0, t0, done
        slli s0, s0, 1
        addi s1, s1, 1
        j    while
done:
```

Assembly tests opposite case (`pow == 128`) of high-level code (`pow != 128`)

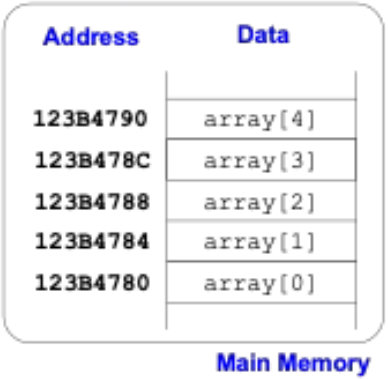
Arrays

- Access large amounts of similar data
- **Index**: access each element
- **Size**: number of elements

Arrays

- 5-element array
- **Base address** = 0x123B4780 (address of first element, array[0])
- First step in accessing an array: load base address into a register

Accessing Arrays




```
// C Code
   int array[5];
   array[0] = array[0] * 2;
   array[1] = array[1] * 2;

# RISC-V assembly code
# s0 = array base address
  lui   s0, 0x123B4          # 0x123B4 in upper 20 bits of s0
  addi  s0, s0, 0x780            # s0 = 0x123B4780

  lw    t1, 0(s0)            # t1 = array[0]
  slli  t1, t1, 1            # t1 = t1 * 2
  sw    t1, 0(s0)            # array[0] = t1

  lw    t1, 4(s0)            # t1 = array[1]
  slli  t1, t1, 1            # t1 = t1 * 2
  sw    t1, 4(s0)            # array[1] = t1
```

Accessing Arrays Using For Loops

```
// C Code
   int array[1000];
   int i;

   for (i=0; i < 1000; i = i + 1)
    array[i] = array[i] * 8;

# RISC-V assembly code
# s0 = array base address, s1 = i
```

Accessing Arrays Using For Loops

```
// C Code
    int array[1000];
    int i;

    for (i=0; i < 1000; i=i+1)
      array[i] = array[i] * 8;
```

```
# RISC-V assembly code
# s0 = array base address, s1 = i
# initialization code
  lui  s0, 0x23B8F          # s0 = 0x23B8F000
  ori  s0, s0, 0x400        # s0 = 0x23B8F400
  addi s1, zero, 0          # i = 0
  addi t2, zero, 1000       # t2 = 1000

loop:
  bge  s1, t2, done         # if not then done
  slli t0, s1, 2            # t0 = i * 4 (byte offset)
  add  t0, t0, s0           # address of array[i]
  lw   t1, 0(t0)            # t1 = array[i]
  slli t1, t1, 3            # t1 = array[i] * 8
  sw   t1, 0(t0)            # array[i] = array[i] * 8
  addi s1, s1, 1            # i = i + 1
  j    loop                 # repeat
done:
```

Check-In Activity

With a group, see the handout:

1. *Copy the previous assembly example into the Venus simulator, step through / run the program, answering the following questions:*
    - *What is the value of s1 at the end of the execution of the program, and why?*
    - *What is the value of t1 at the end of the program, and why?*
2. *Write an assembly program that calculates the sum of the values in a 10-element array of ints.* If you're not sure how to start, try these steps:
    - Write the C syntax first, to guide you
    - Recreate the for-loop initialization and conditions from previous example
    - Paste into Venus simulator to debug/test!

Accessing Arrays Using For Loops

```
// C Code
    int array[1000];
    int i;

    for (i=0; i < 1000; i=i+1)
      array[i] = array[i] * 8;
```

```
# RISC-V assembly code
# s0 = array base address, s1 = i
# initialization code
  lui  s0, 0x23B8F          # s0 = 0x23B8F000
  ori  s0, s0, 0x400        # s0 = 0x23B8F400
  addi s1, zero, 0          # i = 0
  addi t2, zero, 1000       # t2 = 1000

loop:
  bge  s1, t2, done         # if not then done
  slli t0, s1, 2            # t0 = i * 4 (byte offset)
  add  t0, t0, s0           # address of array[i]
  lw   t1, 0(t0)            # t1 = array[i]
  slli t1, t1, 3            # t1 = array[i] * 8
  sw   t1, 0(t0)            # array[i] = array[i] * 8
  addi s1, s1, 1            # i = i + 1
  j    loop                 # repeat
done:
```

Multiplication

32 × 32 multiplication ⟶ 64 bit result

`mul s3, s1, s2`

`s3` = lower 32 bits of result

`mulh s4, s1, s2`

`s4` = upper 32 bits of result, treats operands as signed

`{s4,s3} = s1 x s2`

Example: s1 = 0x40000000 = 2^{30}; s2 = 0x80000000 = -2^{31}

s1 x s2 = -2^{61} = 0xE0000000 00000000

s4 = 0xE0000000; s3 = 0x00000000

Division

32-bit division ⟶ 32-bit quotient & remainder

- `div  s3, s1, s2  # s3 = s1/s2`
- `rem  s4, s1, s2  # s4 = s1%s2`

Example: s1 = 0x00000011 = 17; s2 = 0x00000003 = 3

s1 / s2 = 5

s1 % s2 = 2

s3 = 0x00000005; s4 = 0x00000002

Wrap-Up October 29

- Coming up next!
  - You need to take time to PRACTICE with many values, simple examples, before we introduce construct and execute more complex RISC-V assembly programs
- Logistics, Reminders
  - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
  - LP Office hours M 9-10:30AM, Th 2:30-4PM
  - Weekly Exercises (including programming exercises) will be posted Friday, due next week
  - Lab 5 - First Stage as Pre-Lab for next week **individually**
  - Lab 5 Report due November 4 10PM
- FEEDBACK
  - https://forms.gle/5Aafcm3iJthX78jx6