ASSEMBLY PROGRAMMING

LECTURE 08-1

JIM FIX, REED COLLEGE CS2-F20

COURSE LOGISTICS

- ***TODAY:*** processor-level programming
  - Finish working with the MINICS2 circuit/processor and its instruction set.
  - Start looking at MIPS32 programming.
    - ➡ We'll use the **`spim`** simulator to run MIPS32 assembly programs.
    - ➡ (It's like the LogiSim for processor-level programming.)
    - ➡ Jim's office hours: Monday 4:30-6pm.
- ***TOMORROW:*** lab is cancelled; no lab assignment.
    - ➡ Jim's extra office hours: Tuesday 10-11am, 3:30-4:30pm.
- ***STARTING THIS WEEK:*** "drop-in" tutoring
    - ➡ Monday, Tuesday, Wednesday 7-9pm. Will email Zoom links.

COURSE LOGISTICS

- ***WEDNESDAY:*** "in-class" midterm on C++ programming
  - "closed book," i.e. no notes, programming tools, or on-line resources.
  - 5 or 6 problems; 75 minutes to complete, but should only need about 60.
    - ➡ See the practice midterms for typical problems, scope.
  - Taking the exam:

1. You'll "accept" an assignment to create a private Github repo with a PDF.
2. You'll write your answers to problems on paper.
3. You'll upload photos/scan of your answers to the repo on Github.

  - Work for only 75 minutes, submit shortly after.
  - I'll be available for Slack DMs, over email, and on office hours Zoom link.

COURSE LOGISTICS

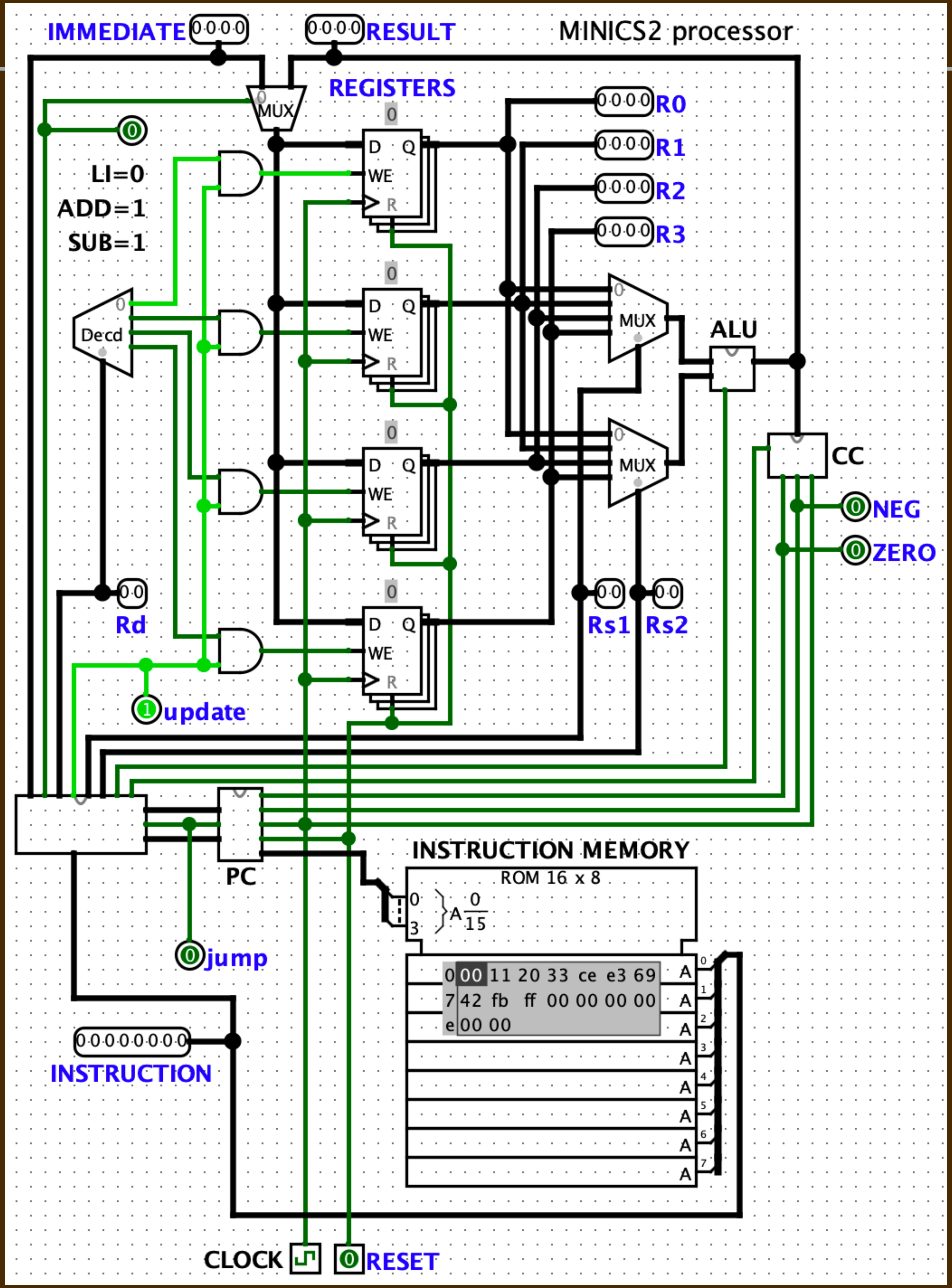
- ***To be posted:***
  - Feedback on Homework 04 and 05. ***TONIGHT***
  - Sample solutions to the practice midterms. ***TONIGHT***
  - Solutions to Homework 04 and 05. ***TOMORROW EVENING***
- This gives you until 7pm to complete/submit Homework 04 and 05 for significant credit.
  - ➡Use office hours to seek help completing that work.

RESPONSIBILITIES, OR LACK THEREOF

- No Homework 07 on sequential circuits. Just Lab 07 for BONUS credit.
- No Lab 08 this week.
- Project 1 *"stats and chats"* due October 30th.
- Lab 09 and Homework 09 next week will cover MIPS32 programming.

MINICS2 PROCESSOR

- Four 4-bit registers
  - named $R0-$R3
- 16-byte program memory
  - holds sequence of 8-bit instructions
  - load, add, subtract, compare registers.
  - can (conditionally) jump (i.e. "branch")
- Two "program state" registers
  - PC: program counter
    - ➡ which instruction is executing
  - CC: condition codes
    - ➡ last comparison result NEG or ZERO
- No additional memory.



A MINICS2 PROGRAM

```
#
# A MINICS2 program that sums 1+2+3, storing
# the result in register $R0.
#
L0: LI   $R0, 0        # sum = 0
L1: LI   $R1, 1        # inc = 1
L2: LI   $R2, 0        # count = 0
L3: LI   $R3, 3        # last = 3
L4: CMP  $R3, $R2       # if last - count == 0 go to L9
L5: BCCZ +3             #
L6: ADD  $R2, $R2, $R1  # count += inc
L7: ADD  $R0, $R0, $R2  # sum += count
L8: B    -5             # go to L4
L9: B    -1             # go to L9
```

THAT MINICS2 PROGRAM'S BYTES

```
L#    bits         | hex value
0x0: 00 00 00 00 | 00
0x1: 00 01 00 01 | 11
0x2: 00 10 00 00 | 20
0x3: 00 11 00 11 | 33
0x4: 11 00 11 10 | CE
0x5: 11 10 00 11 | E3
0x6: 01 10 10 01 | 69
0x7: 01 00 00 10 | 42
0x8: 11 11 10 11 | FB
0x9: 11 11 11 11 | FF
```

INTERPRETING THE INSTRUCTION BYTES

```
instruction, mnemonic  ||i76|i54|i32|i10|| meaning
-----------------------++---+---+---+---++----------------------------
load immediate   | LI   || 00|  d| vH  vL|| Rd := v
add              | ADD  || 01|  d| s1| s2|| Rd := Rs1 + Rs2
subtract         | SUB  || 10|  d| s1| s2|| Rd := Rs1 - Rs2
-----------------------++---+---+---+---||
compare (set CC)| CMP  || 11  00| s1| s2|| CC := NZ(Rs1 - Rs2)
-----------------------++---+---+---+---||
branch if neg    | BCCN || 11  01| oH  oL|| if N(CC): PC := PC + o + 1
branch if zero   | BCCZ || 11  10| oH  oL|| if Z(CC): PC := PC + o + 1
branch           | B    || 11  11| oH  oL|| PC := PC + o + 1
-----------------------++---+---+---+---||----------------------------
```

A MINICS2 "ASSEMBLY" PROGRAM

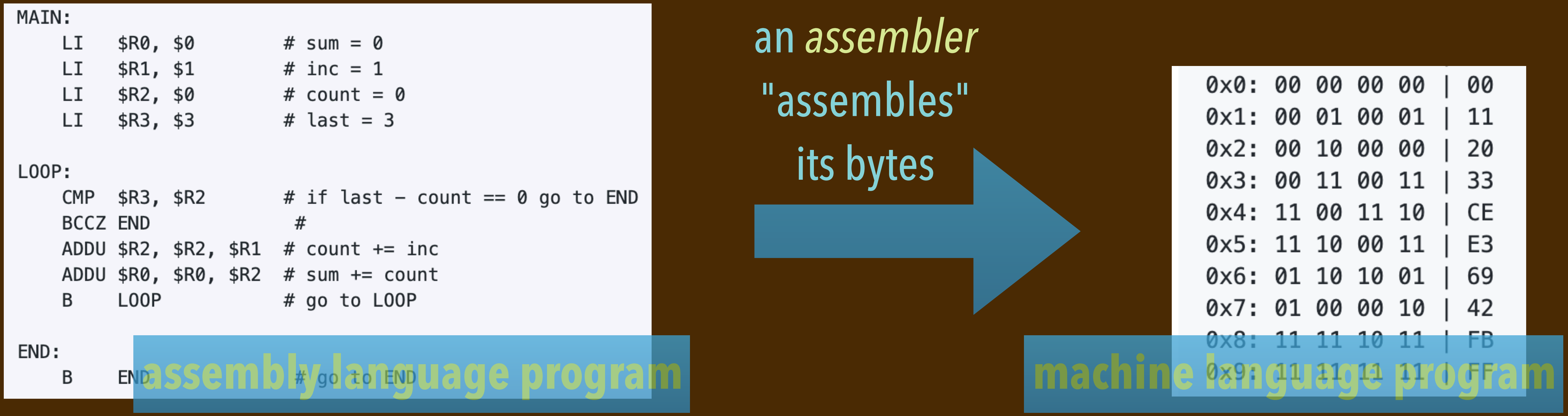
```
#
# A MINICS2 assembly program that sums 1+2+3, storing
# the result in register $R0.
#
MAIN:
    LI   $R0, 0         # sum = 0
    LI   $R1, 1         # inc = 1
    LI   $R2, 0         # count = 0
    LI   $R3, 3         # last = 3

LOOP:
    CMP  $R3, $R2       # if last - count == 0 go to END
    BCCZ END            #
    ADD  $R2, $R2, $R1  # count += inc
    ADD  $R0, $R0, $R2  # sum += count
    B    LOOP           # go to LOOP

END:
    B    END            # go to END
```

ASSEMBLY LANGUAGE PROGRAMMING

- An *assembly language* (or *assembler language*) *program* is a human-readable "processor-level" program written using "mnemonic" instructions from that processor's language.
- A *machine-language program* is the actual sequence of bytes of the program's binary image.
- Uses easier-to-read elements: labels, constants, register names, mnemonics.



```
MAIN:
    LI   $R0, $0         # sum = 0
    LI   $R1, $1         # inc = 1
    LI   $R2, $0         # count = 0
    LI   $R3, $3         # last = 3

LOOP:
    CMP  $R3, $R2        # if last - count == 0 go to END
    BCCZ END              #
    ADDU $R2, $R2, $R1   # count += inc
    ADDU $R0, $R0, $R2   # sum += count
    B    LOOP            # go to LOOP

END:
    B    END             # go to END
```

assembly language program

an *assembler* "assembles" its bytes

```
0x0: 00 00 00 00 | 00
0x1: 00 01 00 01 | 11
0x2: 00 10 00 00 | 20
0x3: 00 11 00 11 | 33
0x4: 11 00 11 10 | CE
0x5: 11 10 00 11 | E3
0x6: 01 10 10 01 | 69
0x7: 01 00 00 10 | 42
0x8: 11 11 10 11 | FB
0x9: 11 11 11 11 | FF
```

machine language program

MINICS2 INSTRUCTION: LOAD IMMEDIATE

- Load an "immediate value" into a destination register.

  Mnemonic code: `LI $R`*dest*`,` *value*

  - *dest* is one of 0, 1, 2, 3
  - *value* is a 4-bit two's complement-encoded integer

- Instruction format: opcode **`00`**

```
||i76|i54|i32|i10||
++---+---+---+---++
|| 00|  d| vH  vL||
```

- Instruction meaning:

  Rd := v

  PC := PC + 1

MINICS2 INSTRUCTION: ADD

- Sum two source registers; place result into a destination register.

  Mnemonic code: **ADD $R***dest***,$R***src1***,$R***src2*

  - *dest*, *src1*, *src1* are each one of 0, 1, 2, 3

- Instruction format: opcode **01**

```
||i76|i54|i32|i10||
++---+---+---+---++
|| 01|  d| s1| s2||
```

- Instruction meaning:

  Rd := Rs1 + Rs2

  PC := PC + 1

MINICS2 INSTRUCTION: COMPARE

- Subtract two source registers; save condition bits; discard result.

  Mnemonic code: `CMP  $R`*src1*`,$R`*src2*

  - *src1*, *src1* are each one of 0, 1, 2, 3

- Instruction format: opcode **`1100`**

```
||i76|i54|i32|i10||
++---+---+---+---++
|| 11| 00| s1| s2||
```

- Instruction meaning:

  Ncc := isNegative(Rs1 - Rs2)

  Zcc := isZero(Rs1 - Rs2)

  PC := PC + 1

MINICS2 INSTRUCTION: BRANCH ON RESULT OF ZERO

- Jump to a labelled instruction if last comparison resulted in zero (set ***Zcc***).

  Mnemonic code: `BCCZ` *label*

- Instruction format: opcode **`1110`**

```
||i76|i54|i32|i10||
++---+---+---+---++
|| 11| 10| oL  oH||
```

- Instruction meaning:

  if ***Zcc = 1*** then ***PC := PC + 1 - o*** else ***PC := PC + 1***

NOTE: `CMP` and `BCCZ` are like C++ "`if (Rs1 == Rs2) { ... }`"

MINICS2 INSTRUCTION: BRANCH ON NEGATIVE RESULT

- Jump to a labelled instruction if last comparison resulted in zero (set ***Ncc***).

  Mnemonic code: `BCCN` *label*

- Instruction format: opcode **`1101`**

```
||i76|i54|i32|i10||
++---+---+---+---++
|| 11| 01| oL  oH||
```

- Instruction meaning:

  if $Ncc = 1$ then $PC := PC + 1 - o$ else $PC := PC + 1$

NOTE: `CMP` and `BCCN` are like C++ "`if (Rs1 < Rs2) { ... }`"

MIPS32 PROCESSOR

- See https://en.wikipedia.org/wiki/MIPS_architecture_processors
- Thirty-two 32-bit registers.
  - named $v0-$v1, $a0-a3, $t0-$t9, $s0-$s7, $fp, $sp, $ra, a few others (some are *reserved*)
- Instructions are 32 bits wide.
- In addition to registers, processor typically has access to an addressable "random access" memory (RAM)
  - combined program/data
  - readable/writeable
  - addresses are 32 bits wide.

SAMPLE MIPS32 ASSEMBLY PROGRAM

```
    .globl main
    .text

main:
    li   $t0, 0          # sum = 0
    li   $t1, 1          # inc = 1
    li   $t2, 0          # count = 0
    li   $t3, 100        # last = 100
loop:
    beq  $t3, $t2, done  # if last == count goto done
    add  $t2, $t2, $t1   # count += inc
    add  $t0, $t0, $t2   # sum += count
    b    loop

done:
    li   $v0, 0          # return 0
    jr   $ra             #
```