Assembly Language

Introduction to Basic Control Structures Written in Assembly

By Jason Agron

Program Control Structures?

- Things like…
  - Conditional Statements.
    - IF Statements.
    - CASE Statements.
  - LOOP Statements:
    - DO-WHILE.
    - WHILE.
    - FOR.

Conditional Statements

- Used to “test” a condition.
  - Check if it is TRUE or FALSE.
- Implemented using…
  - A comparison.
  - A branch (or many branches).
  - A body of code to execute if TRUE.
  - A body of code to execute if FALSE.

Example Introduction

- Example:

  if (x == 0) then

  <bodyTrue>

  else

  <bodyFalse>

  end if

- Notes:
- *"then"*
  - If true, fall through.
  - If false, jump to *<bodyFalse>*.
- *<bodyTrue>*
  - Last statement must be followed by a jump to *"end if"*.

IF Statement Example

- Pseudo-code:

```
if (x == 0) then
    <bodyTrue>
else
    <bodyFalse>
end if
<restOfProgram>
```

NOTE: Assume x is stored in r8.

- MB Assembly:

```
        # Check Condition…
        # Either fall through to bodyTrue
        # Or, branch to bodyFalse
        bneqi r8, 0, bodyFalse
        nop
bodyTrue:
        <bodyTrue>
        # Jump to endIf
        bri endIf
        nop
bodyFalse:
        <bodyFalse>
endIf:
        <restOfProgram>
```

CASE Statements

- Just like if statements, but…
  - Extra comparisons need to be made.
    - i.e. ELSE-IFs.
- Try to write assembly for the following:

case(x):

when "0": *<body0>*

when "1": *<body1>*

default: *<bodyD>*

end case

LOOP Statements

- Just like IF statements, but…
  - They have a “backward” branch.
- Implemented using…
  - A condition:
    - Do I continue and enter the loop or do I exit the loop.
  - The loop body.
  - The backwards branch.
    - To repeat, and re-check the loop condition.

Example Introduction

- Example:

```
x = 0
while (x < 99) {
    x = x + 2
    <loopBody>
}
```

- Notes:
  - Comparison could either fall through into loop OR jump to end of loop.
  - Last statement in <*loopBody*> should jump back to comparison.

WHILE Loop Example

- Pseudo-code:

```
x = 0
while (x < 99) {
    x = x + 2
    <loopBody>
}
<restOfProgram>
```

NOTE: Assume x is stored in r8.

- MB Assembly:

```
        # Init x to 0
        addi r8 r0 0
loopCond:
        # Check condition
        bgti r8, 99, endLoop
loopBody:
        addi r8, r8, 2
        <loopBody>
        bri loopCond
        nop
endLoop:
        <restOfProgram>
```

MB Assembly

- MicroBlaze ISA Documentation:
  - http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf
- Additional assembly examples:
  - Can be found on the web.
    - OR
  - Can be developed by working with your TA.

Why Learn Assembly?

- To learn an ISA is to understand the relationship between high-level languages and the machines that "execute" them.
- Also, some CPUs do not have compilers for them, therefore knowing the ISA would allow one to…
  - Program the CPU.
  - Write a compiler for the CPU.
  - Etc.

Goal of This Lab

- Re-implement your GPIO-based FSM in assembly.
  - Make sure to document your programs!!!!
- It is allowed to have a “simpler” behavior, but it must…
  - Have at least 4 distinct “states”.
  - One of the states must be “dynamic”.
    - LEDs must flash in some sort of alternating pattern.

Questions

1) What does ISA stand for?
2) What are R0 and R1 used for in the MB?
3) How many bits are used for the *opcode* in all MB instructions?
4) Write individual programs using the MB's ISA that…
   a) Adds R4 and R5 and stores the result in R9.
   b) Stores the lower byte in R3 at memory location 0xDEADBEEF.
   c) Zero out all of the bits of R10 except the lower byte and store the result in R10.