Lab RISC-V to WASM

CMPUT 229

Your task in this lab

- To generate the function body of a WASM module from a RISC-V function.

- You are provided with some components of the WASM module so that the WASM code that you generate can run in a browser.

What is a WASM module?

WASM Module

- A file containing a webassembly program
- The smallest valid WASM module contains 8 bytecodes:

```
0x00, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00
```

- WAST is a text representation:

```
(module)
This module does not contain any code!
```

WASM Module Sections

- Sections have headers and headers have fields.

- All sections are optional
  - At least a few sections and fields in each header should be included to do something interesting.

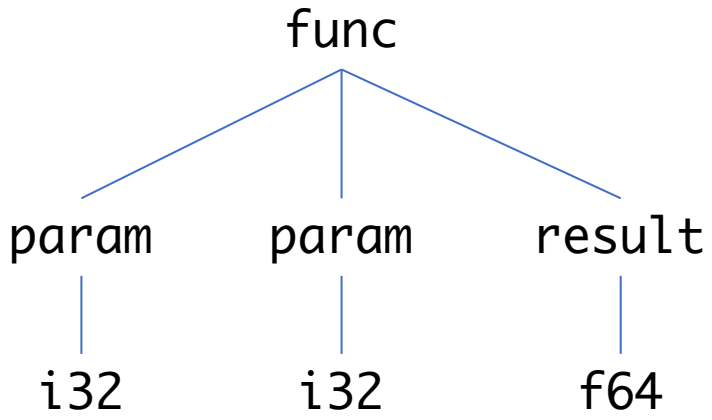
- The lab provides you with:
  - a type section,
  - a function (declaration) section,
  - an export section, and
  - the preamble for the code section.

Main Idea -RISCVtoWASM

- Parse the binary representation of a RISC-V program discovering branches:
  - Compute each branch target
  - Increment either the backward or the forward branch of the target

- Parse the binary representation of a RISC-V program again:
  - Detect if the instruction is an I-type, R-type, or forward/backward branch instruction and then build a set of corresponding WASM instructions
  - Place the WASM instructions into the provided output space
- Branch translation is special and different for forward and backward branches

S-Expression

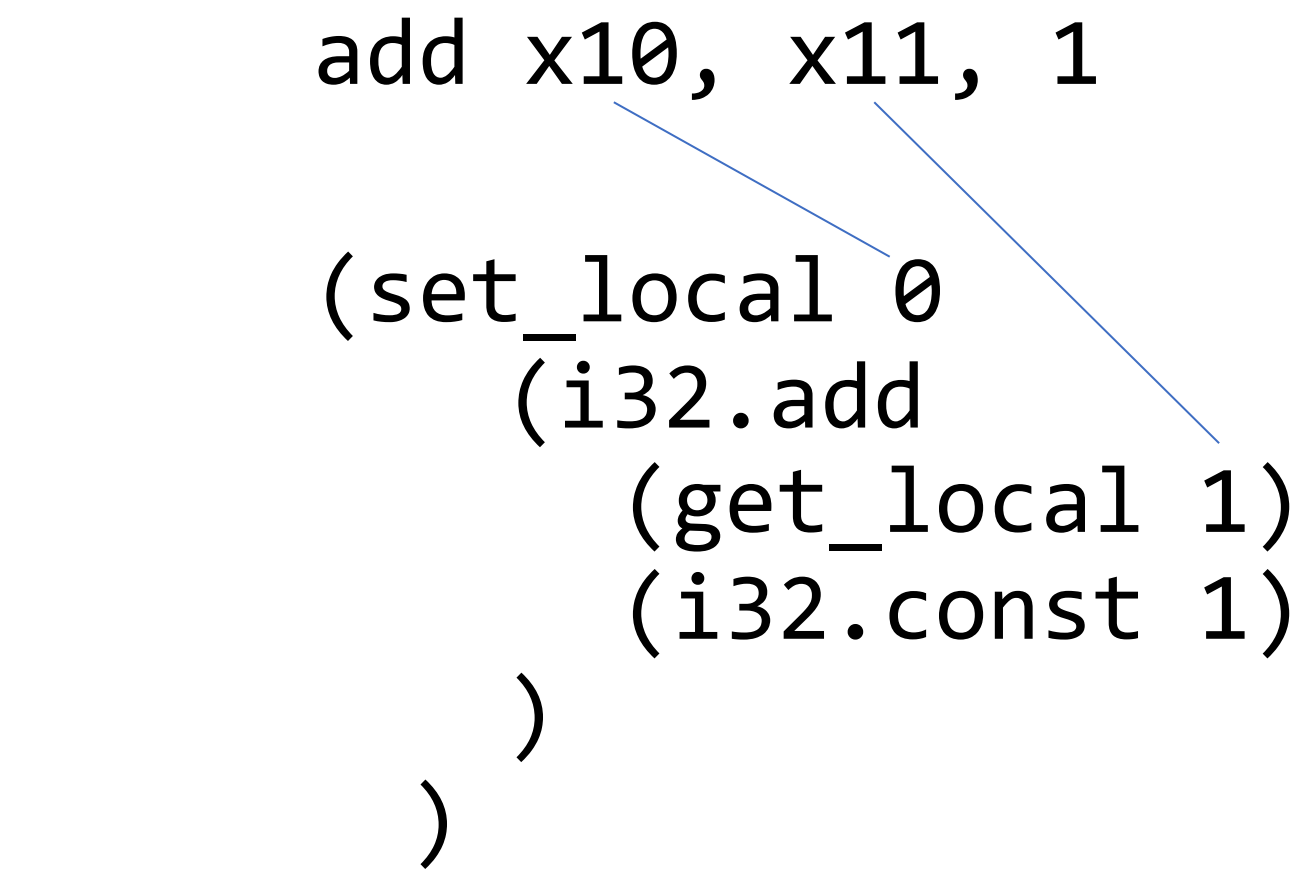


(func (param i32) (param i32) (result f64) ...)

The lab provides a template S-expressions that you can use to generate bytecode.

Translating RISC-V to WAST

(Webassembly text representation)


add x10, x11, 1
(set_local 0
(i32.add
(get_local 1)
(i32.const 1)
)
)

```
add x10, x11, x12


(set_local 0
    (i32.add
      (get_local 1)
      (get_local 2)
    )
  )
```

```
add x10 x11 x0


(set_local 0
    (i32.add
      (get_local 1)
      (i32.const 0)
    )
  )
```

When encountering the zero register you must change it to a constant of zero

```
        …
        beq x10 x11 label
        add x11 x21 x20
        add x10 x11 x12   A
label:
        add x10 x11 x12   B
        …
```

```
(block
    (i32.eq (get_local 0) (get_local 1) )
    br_if 0
    (⟨instructions in A⟩)
end)
(⟨instructions in B⟩)
```

```
        …
label:
        add x11 x21 x20
        add x10 x11 x12
        beq x10 x11 label
        add x10 x11 x12
        …
```

A

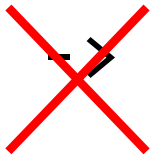
```
(loop
    (⟨instructions in A⟩)
    (i32.eq (get_local 0) (get_local 1))
    br_if 0
end)
```

S-expression to Bytecode

- When determining the bytecode ordering in the module one cannot simply transcribe components of the S-expression representation:

- Incorrect translation:

```
(set_local x0 (i32.const 1))
```



```
22   ;set_local
00   ;variable index
41   ;i32.const
01   ;literal value
```

Bytecode ordering

- WASM module uses postfix notation.
  - operators follow the operands in an expression.

- Examples of WAST on the web are often shown in prefix notation.

S-expression to Bytecode

- Correct translation:

```
(set_local x0 (i32.const 1))     ->   41   ;i32.const
                                      01   ;literal value
                                      22   ;set_local
                                      00   ;variable index
```

- Bytecode for the `x0` the constant (i32.const 1) operands appear before the set_local operator.

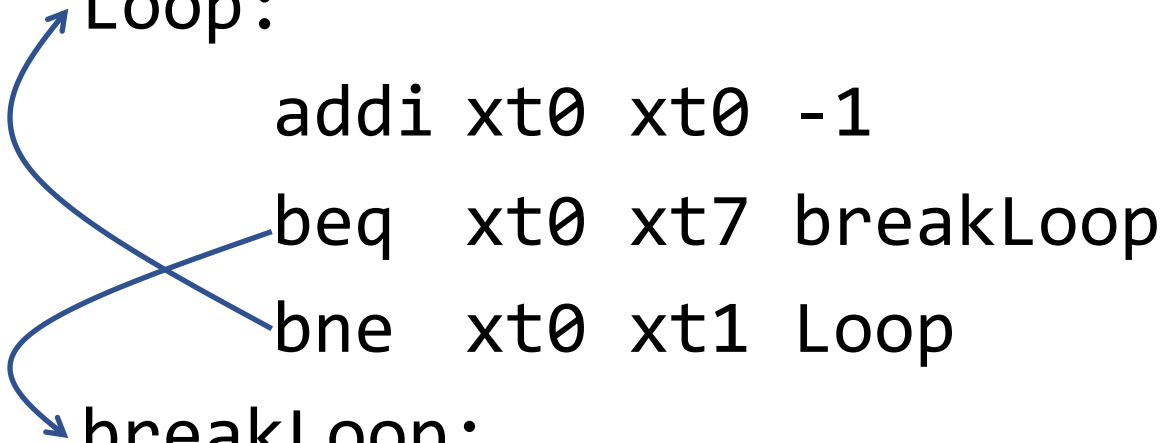
Control Flow

- WASM enforces structured control flow in a program. In RISC-V however it is possible to have unstructured control flow.

- Translation a RISC-V unstructured program results in either invalid or nonsense WASM code
  - All testcases must be structured code.

Unstructured Example

- Multiple exits/entry points out of a loop is unstructured control flow

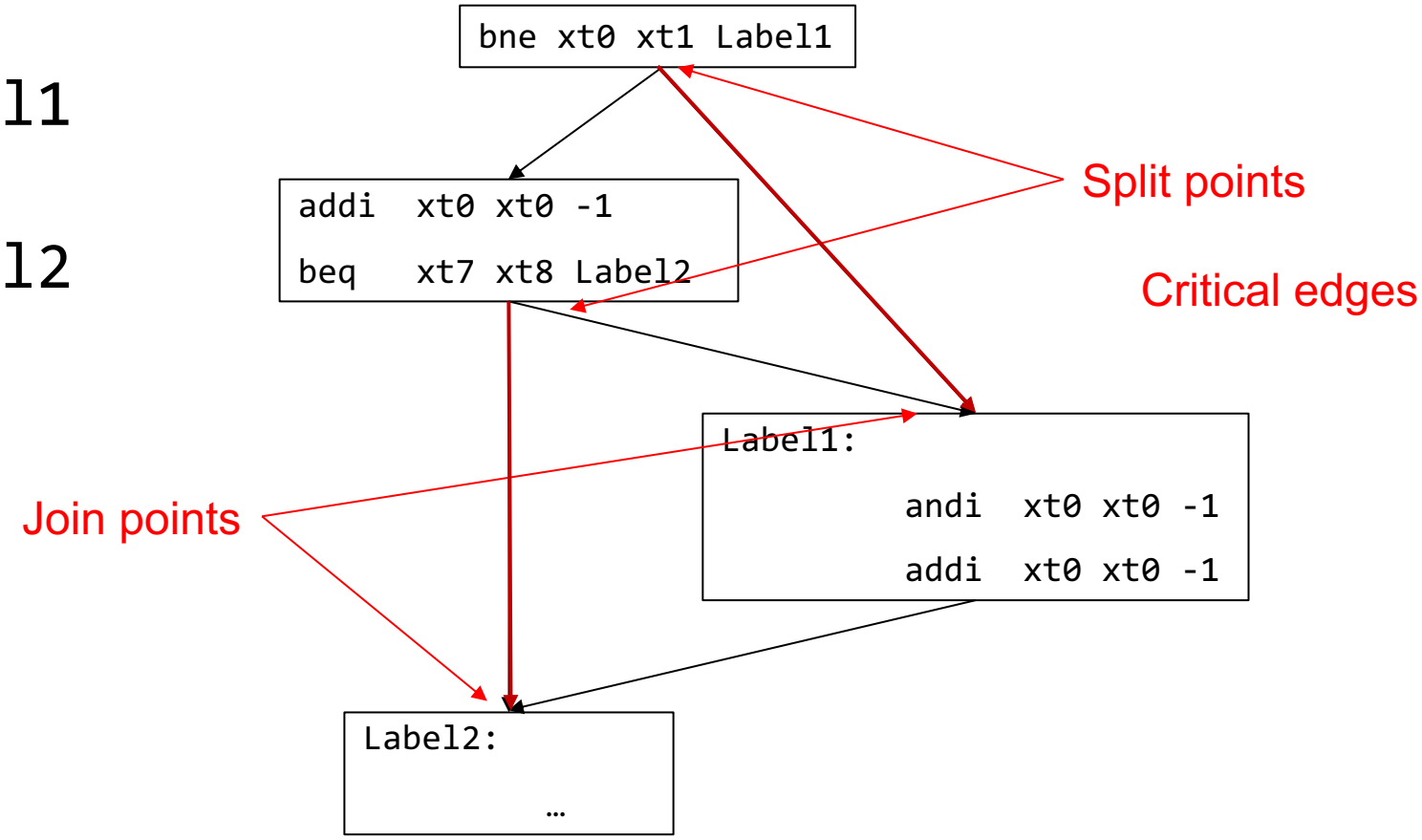


```
Loop:
        addi xt0 xt0 -1
        beq  xt0 xt7 breakLoop
        bne  xt0 xt1 Loop
breakLoop:
        …
```

Unstructured Example

- Branching to the instruction immediately after another branch creates unstructured code

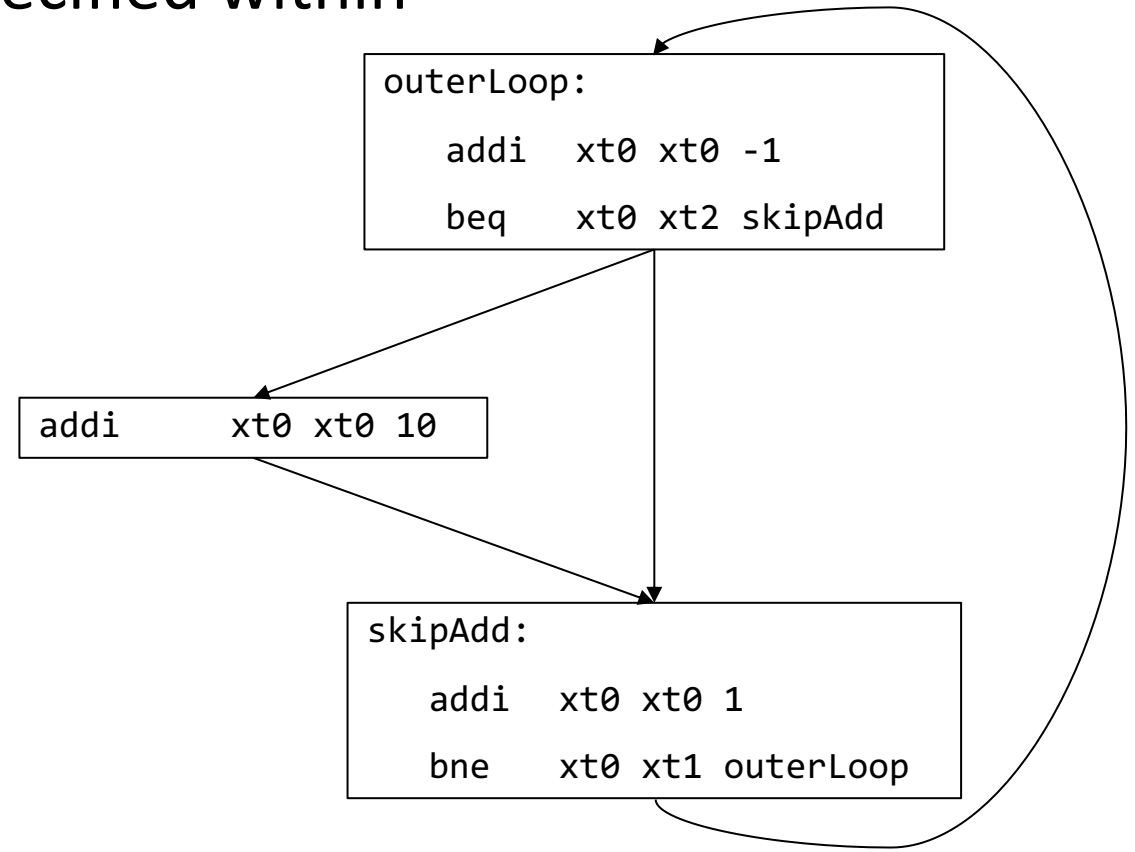
```
        bne  xt0 xt1 Label1
        addi xt0 xt0 -1
        beq  xt7 xt8 Label2
Label1:
        andi xt0 xt0 -1
        addi xt0 xt0 -1
Label2:
        …
```



Structured Example

- The control flow of the program can be modeled with fully nested blocks with no target specified within another construct's block.

```
outerLoop:
      addi xt0 xt0 -1
      beq  xt0 xt2 skipAdd
      addi xt0 xt0 10
skipAdd:
      addi xt0 xt0 1
      bne  xt0 xt1 outerLoop
```



LEB128 encoding

- LEB128 encoding compresses a binary representation to use as few bytes as possible
- The number of bytes needed to represent a certain value depends on the value
- For example: 1 can be represented in 1 byte, but 128 needs 2 bytes

LEB128 Example

What is the into LEB128 format for 321345?

321345 = 1001110011101000001 (19 bits)

Sign extend to a multiple of 7 bits

321345 = 001001110011101000001 (now 21 bits)

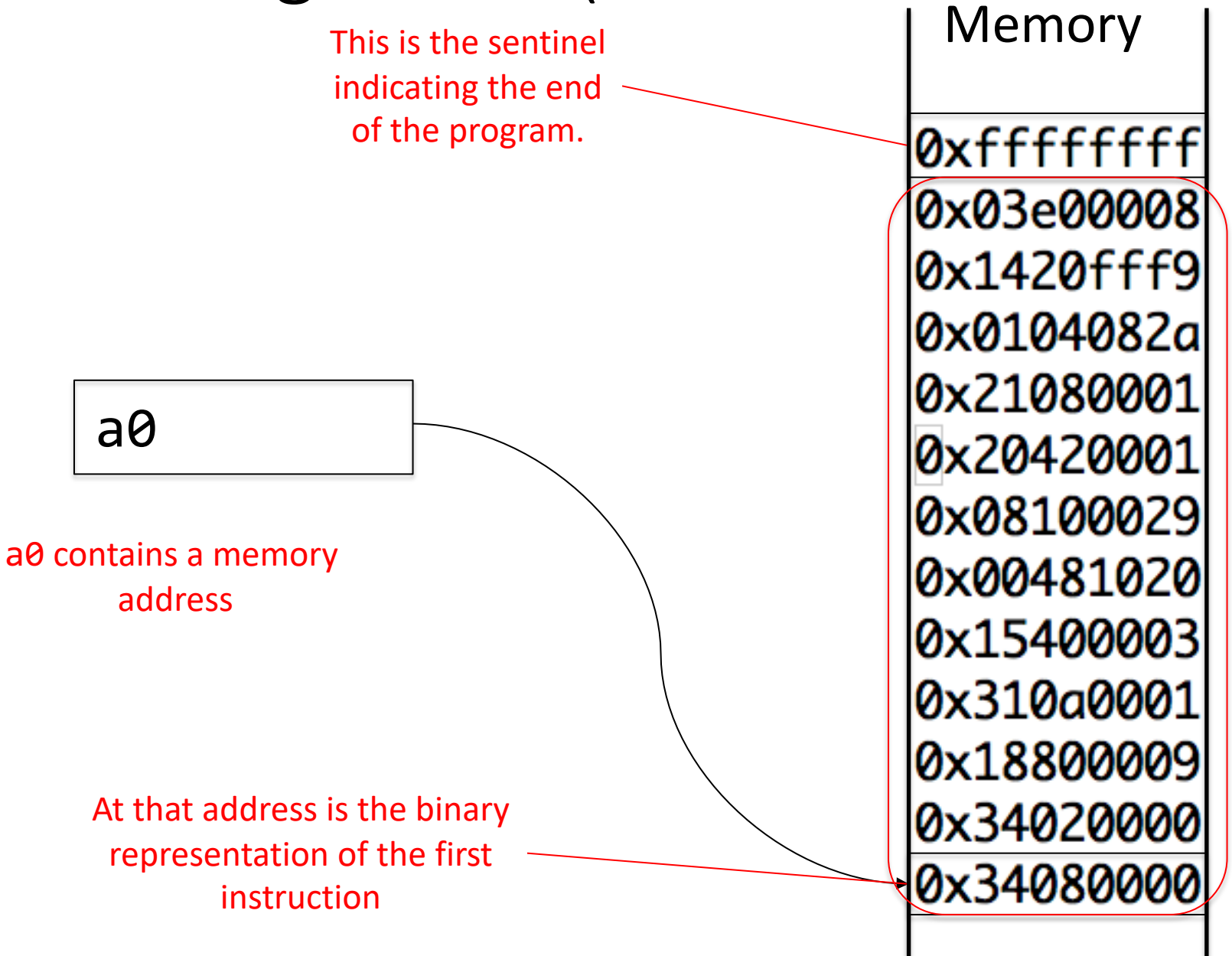
Break into groups of 7 bits

Set the highest bit on each byte to 1, except for most significant.

00010011 11001110 11000001

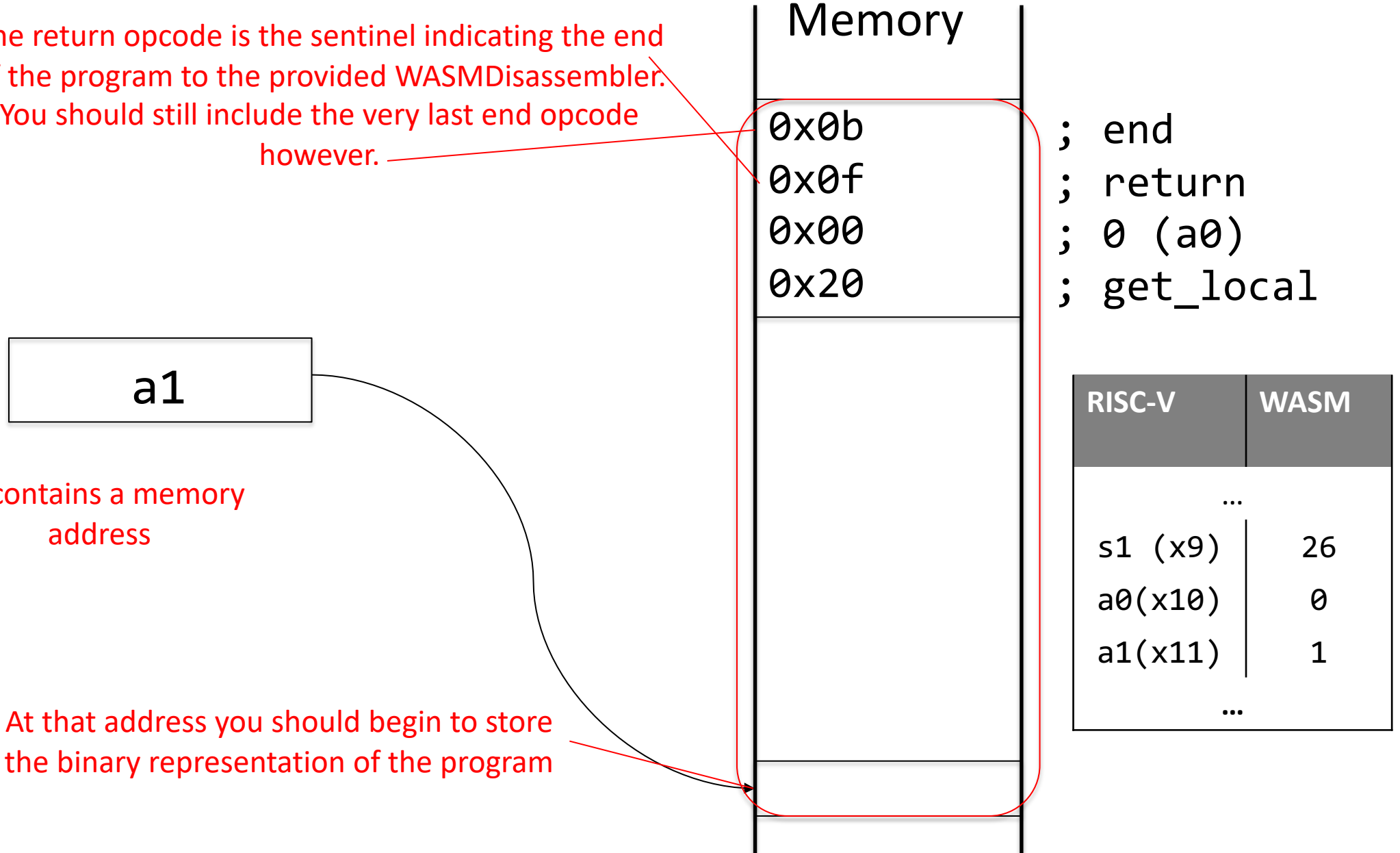
0x13 0xce 0xc1

The Assignment (RISCVtoWASM input)



The Assignment (RISCVtoWASM input)

The return opcode is the sentinel indicating the end of the program to the provided WASMDisassembler. You should still include the very last end opcode however.



The Assignment (RISCVtoWASM input)

`a0` – number of bytes that have been generated to represent the generated WASM program.

RISCVtoWASM **Effect:**
The address provided to you in `a1` will point to the space that should now contain your generated WASM code.

Testing

- Test Cases
  - One test case is provided, under the link in the assignment specification
  - The browserTool directory also contains sample RISC-V functions as well as their respective .bin files to be used as input to your program and the .wasm files that are the translated result.
- Student-Generated Test Cases
  - Students will submit test cases
- Printing the Output of your solution
  - RISCV code provided for printing

Tips – Overall:

- Test as you go, there is a disassembler provided so that you may analyze intermediate output. The only requirement to run the disassembler is that your code is **terminated with a return statement**.
- You may also use the `hexdump` command to view your program's binary file (.wasm file) output
- Test your encodeLEB128 function extensively. You may use the decodeLEBtoDec function in the WASMDisassembler.s file to convert your encodeLEB128 output back to decimal format in order to make testing less complex. See the decodeLEBtoDec function comments in the WASMDisassembler.s file for details.
- Start early, this assignment has several nontrivial parts.

University of Alberta
Code of Student Behavior

http://www.governance.ualberta.ca/en/CodesofConductandResidenceCommunityStandards/CodeofStudentBehaviour.aspx

30.3.2(1) Plagiarism

No Student shall submit the words, ideas, images or data of another person as the Student's own in any academic writing, essay, thesis, project, assignment, presentation or poster in a course or program of study.

30.3.2(2) Cheating

30.3.2(2) d No Student shall submit in any course or program of study, without the written approval of the course Instructor, all or a substantial portion of any academic writing, essay, thesis, research report, project, assignment, presentation or poster for which credit has previously been obtained by the Student or which has been or is being submitted by the Student in another course or program of study in the University or elsewhere