CS61C Spring 2025

Precheck: Inst. Trans., AMAT

Discussion 5

1 Discussion Pre-Check

1.1 True or False: In RISC-V, the opcode field of an instruction determines its type (R-Type, S-Type, etc.).

Answer: True.

The opcode field of an instruction uniquely identifies the instruction type and allows us to identify the instruction format we're working with.

1.2 Convert the following RISC-V registers into their binary representation:

s0: Looking at the 61C reference sheet, we can see that s0 refers to the x8 register. To get the final answer, we convert 8 into binary: `0b01000`.

Following the same procedure as above, we get the rest of the answers...

sp: `x2 = 0b00010`

x9: `x9 = 0b01001`

t4: `x29 = 0b11101`

1.3 True or False: In RISC-V, the instruction `li x5 0x44331416` will always be encoded in 32 bits when translated into binary.

Answer: False.

This is a bit of a trick question. It is true that every regular instruction in RISC-V will always be encoded in 32-bits. However, li is actually a pseudo-instruction! Recall that pseudo-instructions can translate into one or more RISC-V instructions. In this case, li will be translated into an addi and lui instruction. Therefore, `li x5 0x44331416` will actually be encoded in 64-bits, as it represents two RISC-V instructions.

1.4 True or False: We can use a branch instruction to move the PC by one byte.

Answer: False

Branch instruction offsets have an implicit zero as the least significant bit, so we can only move the PC in offsets divisible by 2 (refer back to Lecture 13 for an explanation why this is!).

The full offset for a branch instruction will be the 13-bit offset `{imm[12:1], 0}`, where we take the immediate bits from our instruction's binary encoding and add the implicit zero.

2 Instruction Translation

Recall that every instruction in RISC-V can be represented as a 32-bit binary value, which encodes the type of instruction, as well as any registers/immediates included in the instruction. To convert a RISC-V instruction to binary, and vice-versa, you can use the steps below. The 61C reference sheet will be very useful for conversions!

RISC-V ⟹ Binary

(a) Identify the instruction type (R, I, I*, S, B, U, or J)
(b) Find the corresponding instruction format
(c) Convert the registers and immediate value, if applicable, into binary
(d) Arrange the binary bits according to the instruction format, including the opcode bits (and possibly funct3/funct7 bits)

Binary ⟹ RISC-V

(a) Identify the instruction using the opcode (and possibly funct3/funct7) bits
(b) Divide the binary representation into sections based on the instruction format
(c) Translate the registers + immediate value
(d) Put the final instruction together based on instruction type/format

Below is an example of a series of RISC-V instructions with their corresponding binary translations.

example.S	example.bin

`main:`
`  addi  sp,sp,-4`
`  sw  ra,0(sp)`
`  addi  s0,sp,4`
`  mv  a0,a5`
`  call  printf`
`  ...`
	`...`
	`11111111110000010000000100010011`
	`00000000000010001001000000100011`
	`00000000010000010000010000010011`
	`00000000000000000000010100010011`
	`00000000010001000000000011101111`
	`...`

3 AMAT (Average Memory Access Time)

Recall that AMAT stands for Average Memory Access Time. This is a way to measure the performance of a cache system. The formula for AMAT is:

AMAT = (Hit Time) + (Miss Rate) * (Miss Penalty)

In a multi-level memory hierarchies (e.g. multi-level caches), we can separate miss rates into two types that we consider for each level.

- **Global:** Calculated as the number of accesses that missed at that level divided by the total number of accesses **to the memory system**.
- **Local:** Calculated as the number of accesses that missed at that level divided by the total number of accesses **to that memory level**.