ANNA Programming Card

Opcode	***Op***	***Operands***	***Description***
0000	`add`	*Rd* Rs_1 Rs_2	Two's complement addition: R(*Rd*) ← R(Rs_1) + R(Rs_2)
0001	`sub`	*Rd* Rs_1 Rs_2	Two's complement subtraction: R(*Rd*) ← R(Rs_1) – R(Rs_2)
0010	`and`	*Rd* Rs_1 Rs_2	Bitwise and operation: R(*Rd*) ← R(Rs_1) & R(Rs_2)
0011	`or`	*Rd* Rs_1 Rs_2	Bitwise or operation: R(*Rd*) ← R(Rs_1) | R(Rs_2)
0100	`not`	*Rd* Rs_1	Bitwise not operation: R(*Rd*) ← ~R(Rs_1)
0101	`shf`	*Rd* Rs_1 *Imm6*	Bit shift. The contents of Rs_1 are shifted left (if *Imm6* is positive) or right with zero extension (if *Imm6* is negative). The shift amount is abs(*Imm6*); the result is stored in R(*Rd)*.
0110	`lli`	*Rd Imm8*	The lower bits (7-0) of *Rd* are copied from *Imm8*. The upper bits (15-8) of *Rd* are equal to bit 7 of *Imm8* (sign extension).
0111	`lui`	*Rd Imm8*	The upper bits (15- 8) of *Rd* are copied from Imm8. The lower bits (7-0) of *Rd* are unchanged.
1000	`lw`	*Rd* Rs_1 *Imm6*	Loads word from memory using the effective address computed by adding Rs_1 with the signed immediate: R(*Rd*) ← M[R(*Rs1*) + *Imm6*]
1001	`sw`	*Rd* Rs_1 *Imm6*	Stores word into memory using the effective address computed by adding Rs_1 with the signed immediate: M[R(Rs_1) + *Imm6*] ← R(*Rd*)
1010	`bez`	*Rd Imm8*	Conditional branch – compares *Rd* to zero. If *Rd* is zero, then branch is taken with indirect target of PC + 1 + *Imm8* as next PC. Immediate is a signed value.
1011	`bgz`	*Rd Imm8*	Conditional branch – compares *Rd* to zero. If *Rd* is greater than zero, then branch is taken with indirect target of PC + 1 + *Imm8* as next PC. Immediate is a signed value.
1100	`addi`	*Rd* Rs_1 *Imm6*	Add immediate: R(*Rd*) ← R(Rs_1) + *Imm6*
1101	`jalr`	*Rd* Rs_1	Jumps to the address stored in register *Rd* and stores PC + 1 in register Rs_1.
1110	`in`	*Rd*	Input instruction: R(*Rd*) ← input
1111	`out`	*Rd*	Output instruction: output ← R(*Rd*). If *Rd* is `r0`, halts the processor.
Assembler Directives	`.halt`		Assemble directive that emits an `out` instruction (`0xF000`) that halts the processor.
	`.fill`	*Imm16*	Assembler directive that fills next memory location with the specified value. Immediate is a signed value.

Registers

- Represented by fields *Rd*, Rs_1, and Rs_2.
- A register can be any value from: `r0`, `r1`, `r2`, `r3`, `r4`, `r5`, `r6`, `r7`.
- Register `r0` is always zero. Writes to register `r0` are ignored.

Immediates

- Represented by fields *Imm6*, *Imm8*, and *Imm16*. The number refers to the size of the immediate in bits.
- Immediates are represented using decimal values, hexadecimal values, or labels. Hexadecimal values must start with '`0x`' and labels must be preceded with '`&`'.
- The immediate fields represent an unsigned value. The immediate field for `lui` is specified using a signed value but the sign is irrelevant as the eight bits are copied directly into the upper eight bits of the destination register.
- Labels refer to the address of the label. If a label is used in a branch, the proper PC-relative offset is computed and used as the immediate.

Comments

- A comment begins with a pound sign '`#`' and continues until the following newline.

Labels

- Label definitions consist of a string of letters, digits, and underscore characters followed by a colon. The colon is not part of the label name.
- A label definition must precede an instruction on the same line.
- A label may only be defined once in a program. Only one label is allowed per instruction. The instruction must appear on the same line as the label.

Instruction Formats

Instructions adhere to one of the following three instruction formats:

R-type (`add`, `sub`, `and`, `or`, `not`, `jalr`, `in`, `out`)

15 12	11 9	8 6	5 3	2 0
Opcode	*Rd*	Rs_1	Rs_2	Unused

I6-type (`addi`, `shf`, `lw`, `sw`)

15 12	11 9	8 6	5 0
Opcode	*Rd*	Rs_1	*Imm6*

I8-type (`lli`, `lui`, `bez`, `bgz`)

15 12	11 9	8	7 0
Opcode	*Rd*	Unused	*Imm8*