The S-Machine Instruction Set Architecture

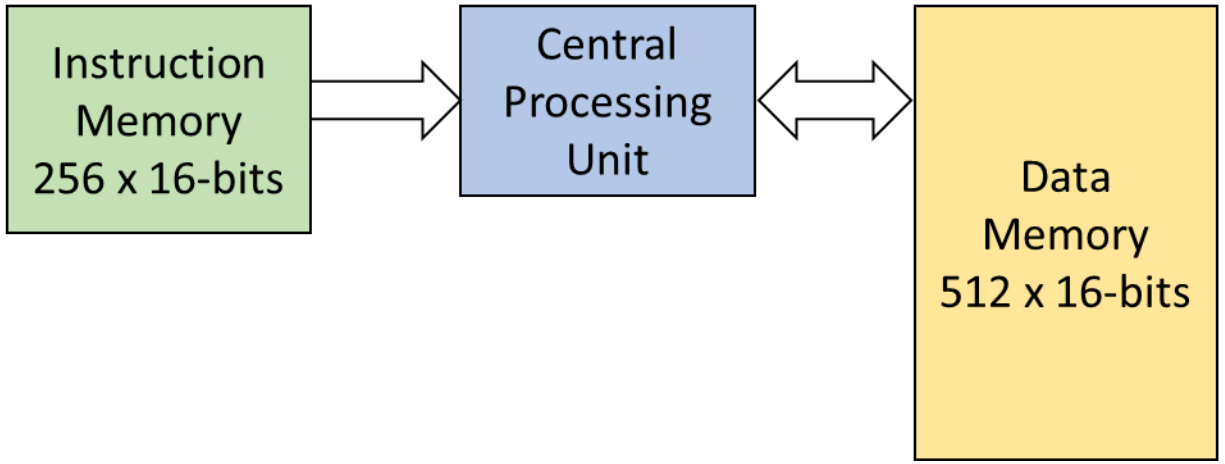
Larry Hughes, PhD
ECED 4260

24 October 2023

Overview

S-Machine's ISA

- 16-bit CPU
- Example of a load-store machine
- Two 16-bit arithmetic registers
- Range of operations (+, -, &, |, and ⊕)
- Up to 256 16-bit instructions
- Up to 512 16-bit data values



Registers (programmer accessible)

- There are three registers that the programmer can access or modify:
- RA – The ‘A’ register. A 16-bit register for arithmetic operations.
  - Always the destination of any operation:

  `RA ← RA <operation> RB`
- RB – The ‘B’ register. A 16-bit register for arithmetic operations.
- PC – Program Counter. An 8-bit register containing the address of the *next* instruction to be executed.
  - PC value at startup is 0x00.
  - PC is incremented by 1 after each instruction fetch.
  - At 0xFF, PC wraps to 0x00.

Status bits

- Three CPU status bits:
- Z: Result of arithmetic operation (RA's result) is zero.
  - All bits are clear (0).
- N: Result of arithmetic operation (RA's result) is negative.
  - *Most significant bit* (<15>) is set (1). All other bits can be set or clear.
- C: Results of arithmetic operation requires 17-bits of storage.
  - 0xFFFF + 0x0001 $\rightarrow$ 0x0000 + C-bit = 1
- Status bits can be *set* or *cleared*.
- Status bits can be used to change the *flow of control*.

Carry bit combinations

RA <15> (Before)	RB <15>	RA <15> (Result)	Carry
0	0	0	0
0	0	1	0
0	1	1	0
0	1	0	1
1	0	1	0
1	0	0	1
1	1	0	1
1	1	1	1

Instruction cycle

Three stage (or phase) instruction cycle

- Fetch
  - The fetch phase fetches the next instruction to be executed, specified by the PC. After the instruction is fetched, the PC is incremented by 1.
- Decode
  - In this phase, the CPU decodes the instruction into is opcode (bits <15:12>) and, depending on the instruction, into its operands. This information is used to setup the execute phase.
- Execute
  - The action associated with the execution cycle depends on the instruction.

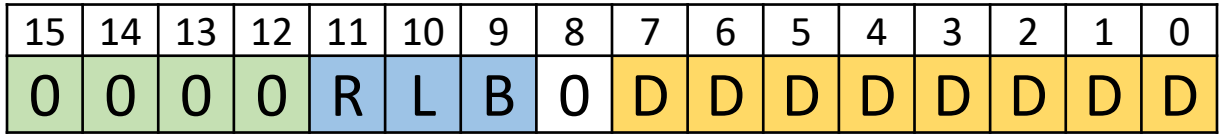
Execute phase

Instruction bits <15:14>	Description
00	Instructions that have operands and 8-bit addresses or data values.
01	Arithmetic and logic instructions without operands.
10	
11	CPU status bit setting and clearing instructions. (CMP is a member of this group, it should be with arithmetic and logic instructions).

Instructions

Memory access (Load)

- LD – Load



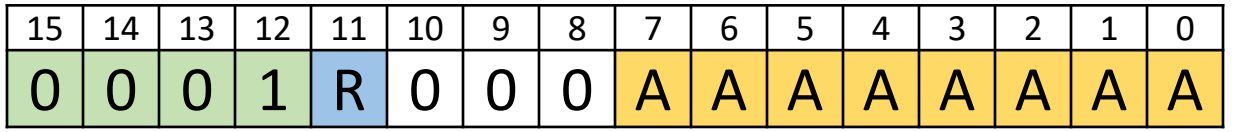
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	R	L	B	0	D	D	D	D	D	D	D	D

- R: Register RA (0) or RB (1)
- L: Location 0: Memory (DDDD.DDDD is address)

  1: Immediate (DDDD.DDDD is an 8-bit value)
- B: Store in low byte (0) or high byte (1)

Memory access (Store)

- ST – Store



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	R	0	0	0	A	A	A	A	A	A	A	A

- R: Register RA (0) or RB (1)
- A: Data memory address

Increment

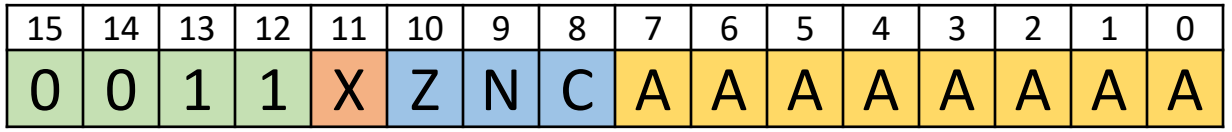
- INC: Add 8-bit immediate value to RA or RB
- Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	R	0	0	0	V	V	V	V	V	V	V	V

- R: Register RA (0) or RB (1)
- V <7:0>: Value to be added to register
- Status bits Z, N, and C can be set or cleared.

Transfer of control

- BR: Conditional or unconditional transfer of control
- Conditional transfer depends on status bit values
- Format:



- X:
  - 0 – Use status as specified (i.e., BZ, branch if Z-bit set)
  - 1 – Invert the status (i.e., *BNZ*, branch if Z-bit not set)
- Z, N, C: Zero, negative, and carry bits to inspect, respectively
- If <11:8> are all set (1) or all clear (0), an unconditional branch occurs

Transfer of control pseudocode

```
Execute ← 0
IF Instruction.bit<11> = 0 THEN
        * BZ, BN, BC plus others
        IF Instruction.bit<10> = Z.bit OR
           Instruction.bit<9> = N.bit OR
           Instruction.bit<8> = C.bit THEN Execute ← 1
        ELSE
        * 0 0 0 0
        IF Instruction.bit<10> = 0 AND
           Instruction.bit<9> = 0 AND
           Instruction.bit<8> = 0 THEN Execute ← 1
        ENDIF
ELSE
        * BNZ, BNN, BNC
        IF (Instruction.bit<10> = 1 AND Z.bit = 0) OR
           (Instruction.bit<9> = 1 AND N.bit = 0) OR
           (Instruction.bit<8> = 1 AND C.bit = 0) THEN Execute ← 1
        ENDIF
        * 1 1 1 1
        IF Instruction.bit<10> = 1 AND
           Instruction.bit<9> = 1 AND
           Instruction.bit<8> = 1 THEN Execute ← 1
        ENDIF
ENDIF
IF Execute = 1 THEN
        PC ← Instruction.bit<7:0>
ENDIF
```

Arithmetic and logic instructions

- Format:

  `RA ← RA <operator> RB`

- Instructions:

Opcode	Operation	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	RA <- RA + RB	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUB	RA <- RA - RB	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
OR	RA <- RA | RB	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
AND	RA <- RA & RB	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
XOR	RA <- RA ⊕ RB	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Status bits are affected

Non-arithmetic instructions

- SHR, MOV, EXCH – affect RA or RB, or both
- CMP – affects Z, N, C status bits
- Format depends on instruction
- Instructions:

Opcode	**Operation**	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SHR	RA <- RA >> 1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
MOV	RB <- RA	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
EXCH	RB <-> RA	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
CMP	Z, N, C <- RA - RB	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Status bits can be affected.

Status bit instructions

- SET and CLR
- Change the status bits
- SET format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	Z	N	C	0	0	0	0	0	0	0	0

- CLR format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	Z	N	C	0	0	0	0	0	0	0	0

SET and CLR pseudocode

- SET

```
IF Instruction.bit<10> = 1 THEN Z.bit ← 1
IF Instruction.bit<9> = 1 THEN N.bit ← 1
IF Instruction.bit<8> = 1 THEN C.bit ← 1
```

- CLR

```
IF Instruction.bit<10> = 1 THEN Z.bit ← 0
IF Instruction.bit<9> = 1 THEN N.bit ← 0
IF Instruction.bit<8> = 1 THEN C.bit ← 0
```