Homework 2

Computer Archiecture and Organization

Sofware Engineering Program,

School of Computer Engineering, KMITL

67011352 Theepakorn Phayonrat

Design Your Own 8-bit Assembly to Hex Instruction Set

Objective:

Design a single instruction set and define a format to represent each instruction to hexadecimal. Then, write sample programs using assembly, and convert them to hex codes suitable for keying into a simulated SBC (Single Board Computer).

System Specifications:

- CPU: 8-bit
- Registers: R0 top R10 (R0 is the accumulator)
- Instruction Size: 3 bytes (24 bits)
  - Byte 1: Opcode
  - Byte 2: Operand 1 (e.g., register)
  - Byte 3: Operand 2 (register, address, or immediate)

Required Instructions:

Mnemonic	Description
LD	Load immediateor from memory
ST	Store accumulator to memory
ADD	Add register or immediate to R0
SUB	Subtract register or immediate
SHL	Shift R0 left by 1 bit
SHR	Shift R0 right by 1 bit
BR	Unconditional branch
BRZ	Branch id zero (R0 == 0)
BRG	Branch id zero (R0 > 0)
JSR	Jump to subroutine
RET	Return from subroutine

You may add 1-2 extra instructions and explain their purpose.

Part 1: Instruction Encoding

1. Define your own opcode mapping. Example:

Mnemonic	Description
LD	0x01
ST	0x02

2. Define instruction format. For example:

LD R0, #12	→	01 00 0C
ADD R0, #20	→	03 00 14

Part 2: Sample Assembly Program

Write an assembly program ($\sim$ 10 instructions) that:

- Load a number into R0
- Add another number
- Stores result in memory
- Checks result and branches if > 0
- Calls a subroutine to clear R0
- Returns to main program

Part 3: Hex Code Conversion

Convert your program into hex. Foer example:

```
LD R0, 12                              ; 01 00 0C
```

Deliverables:

1. Instruction set table with opcodes
2. Assembly program(approx. 10 lines)
3. Hexadecimal representation of program
4. A step-by-step explanation of what each instruction does during the **fetch-decode-execute-store** cycle.
5. Explanation of anny additional instructions

Part 1.0 Answer:

Added 2 instructions

Mnemonic	Description
LD	Load immediateor from memory
ST	Store accumulator to memory
ADD	Add register or immediate to R0
SUB	Subtract register or immediate
SHL	Shift R0 left by 1 bit
SHR	Shift R0 right by 1 bit
BR	Unconditional branch
BRZ	Branch id zero (R0 == 0)
BRG	Branch id zero (R0 > 0)
JSR	Jump to subroutine
RET	Return from subroutine
CMP	Compare 2 registers given as arguments
SYS	System Call (Software Interupt)

Part 1.1 Answer:

Assigned opcode to every instruction

Mnemonic	Opcode
LD	0x01
ST	0x02
ADD	0x03
SUB	0x04
SHL	0x05
SHR	0x06
BR	0x07
BRZ	0x08
BRG	0x09
JSR	0x0A
RET	0x0B
CMP	0x0C
SYS	0x0D

Part 1.2 Answer:

LD R0, #12	$\rightarrow$	01 00 0C
ST R0, #12	$\rightarrow$	02 00 0C
ADD R0, #20	$\rightarrow$	03 00 14
SUB R0, #20	$\rightarrow$	04 00 14
SHL R0, #20	$\rightarrow$	05 00 14
SHR R0, #20	$\rightarrow$	06 00 14
BR R0, #20	$\rightarrow$	07 00 14
BRZ R0, #20	$\rightarrow$	08 00 14
BRG R0, #20	$\rightarrow$	09 00 14
JSR loop	$\rightarrow$	0A 1F (If *loop* subroutine is 31)
RET #1	$\rightarrow$	0B 01
CMP R0, R1	$\rightarrow$	0C 00 01
SYS 1	$\rightarrow$	0D 01

Part 2 Answer:

```
_start:
    LD  R0, =10     @ LD 10 into R0
    ADD R0, #15     @ Add 15 to R0
    ST  R0, [R1]    @ Store result into Memory of R1
    CMP R0          @ Check the value of R0 compare with 0
    BRG b1          @ Branch to b1 if R0 is more than 0
    SWI 0           @ Software interrupt to end the program
clr:
    LD  R0, =0      @ Loads number 0 to R0
    RET  _start     @ Return to _start and rerun until get not greater
```

Part 3 Answer:

```
_start: 01 00 0A
    03 00 0F
    02 00 01
    0C 00
    09 brg
    0D 00
clr:
    01 00 00
    0B _start
```