Sharp R16 Doc

Control Unit

Functional Requirements

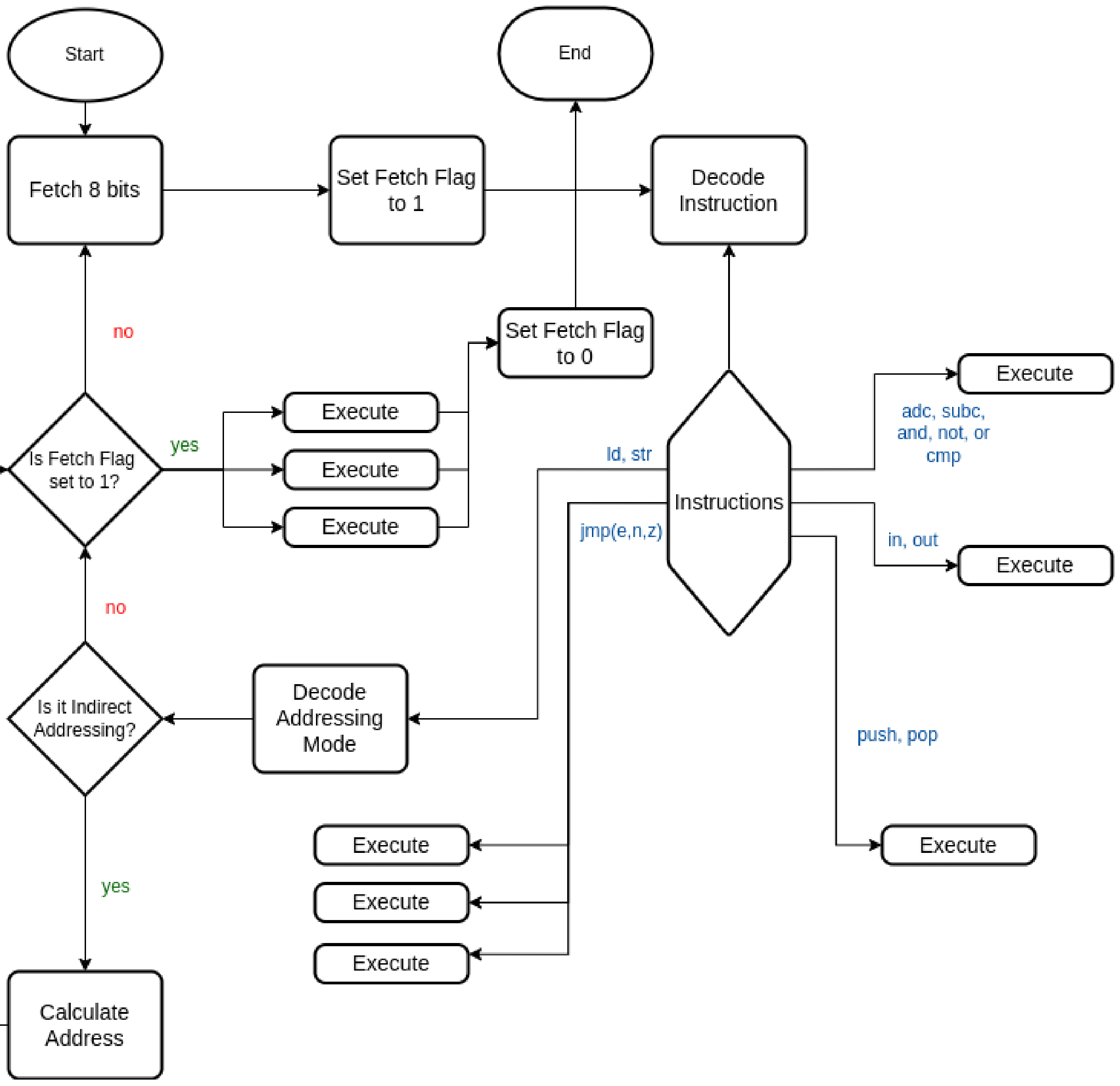
- **Operations**
    1. push - 0000
    2. pop - 0001
    3. adc - 0010
    4. subb - 0011
    5. and - 0100
    6. or - 0101
    7. not - 0110
    8. cmp - 0111
    9. jmpz - 1000
    10. jmpe - 1001
    11. jmpn - 1010
    12. ld - 1011
    13. str - 1100
    14. in - 1101

- **Addressing Modes**
    - depends on the opcode; see Intruction_Format.drawio.png
- **Registers**
    - Register A
    - Register B
    - Register C
    - Register D
- **I/O Module Interface**
    - I/O module to system bus (parallel)
    - I/O module to external device
        - keypad (parallel)
        - LCD module
        - stepper motor (serial)
- **Memory Module Interface**
- **Interrupt Processing Structure**

15. out - 1110

Directives

Instruction Cycle



Machine Cycles

Each instruction is broken down into machine cycles which are in turn, broken down into micro-operations (or states). The number of machine cycles per instruction depends on how many times the CPU needs to access the system bus. Therefore, one machine cycle is defined as the sequence of instructions needed to read/write data from the system bus.

Micro-Operations

Each micro-operation (aka. states) lasts one clock cycle long and multiple micro-operations make up one machine cycle.

Fetch Cycle

*reference: William Stalling’s COA 6th edition

```
t1: MAR <-- (PC)
t2: MBR <-- Memory
        PC <-- (PC) + I
t3: IR <-- (MBR)
```

The addition needed in state t2 can be calculated by the ALU in order to avoid duplicate circuitry

Indirect Cycle

*reference: William Stalling’s COA 6th edition

```
t1: MAR <-- (IR(Address))
t2: MBR <-- Memory
t3: IR(Address) <-- (MBR(Address))
```

Interrupt Cycle

*reference: William Stalling’s COA 6th edition

```
t1: MBR <-- (PC)
t2: MAR <-- Save_Address
        PC <-- Routine_Address
t3: Memory <-- (MBR)
```

Execute Cycle

```
ld
    t1: R <-- Memory
str
    t1: Memory <-- R
adc
    t1: R2 + R1
subb
    t1: R2 - R1
and
    t1: R2 && R1
not
    t1: R2 ! R1
or
    t1: R2 // R1
cmp
    t1: R2 vs R1
jmp(z,e,n) R
    t1: PC <-- R
jmp(z,e,n) immediate
    - run fetch cycle again
    t1: Rjmp <-- immediate
    t2: PC <-- MAR
    t3: MAR - 1 ;subtract 1 because PC would have been increment
                             passed the jmp instruction
    t4: MAR + immediate ;add negative integers to subtract
    t5: PC <-- MAR
push
    t1: stack <-- R
            Rsp + 1
pop
    t1: R <-- stack
            Rsp - 1
in
    t1: Rmain <-- Rgc
```

```
out
    t1: Rgc <-- Rmain
```

Keypad

The keypad includes the following keys:

- A, B, C, D, E, H, I, J, L, N, O, P, R, S, T, Z,
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Execute
- Enter
- Space
- Delete
- left, right, up and down arrows

Keypad to LCD

- Step 1: signal is sent from keypad to its own logic
- Step 2: in logic, the signal is an address in the Character ROM and the corresponding data is read and stored in register ready to be sent to LCD
- Step 3: the data along with its other required control signals is sent to LCD driver to show character
- Step 4: an automatic spacing in the LCD module also appears before the next character can be displayed

LCD

- 5110 Nokia datasheet: https://datasheetspdf.com/pdf-file/1418219/Nokia/5110/1
- Whatever key is pressed on the keypad will generate certain signals for the LCD driver so that the corresponding character will appear.

Source code to LCD

Only when the PRNTR or PRNTI opcodes are encountered will the CPU control the LCD.

- PRINTR
    - Step 1: the CPU's control unit takes the contents of the register and sends it to the 'integer to character decoder' (I might have to be make it myself)
    - Step 2: each character signal sent to the Character ROM as an address
    - Step 3: the data from that address is sent to the LCD driver along with its other required control signals to control the LCD
- PRINTI
    - Step 1: enter the required data inputs for the LCD driver to display the image/character
    - Step 2: the contents is directly sent to the LCD driver along with the control signals to display the image

Memories

The memories in the machine are not memory mapped. All of them are implicitly addressed depending on what the machine needs to do at any given time.

Memory Banks

- Main RAM
- Stack RAM
- Interrupt RAM
- Character ROM
- Assembler ROM
- Look-up table ROM for Assembler
- Program ROM

Registers

- General Purpose (8-bit)

    - 00: GP register
    - 01: GP register
    - 10: GP register
    - 11: GP register
- Graphics Card
    - General Purpose
        - 00: GP register
        - 01: GP register
        - 10: GP register
    - Special Purpose
        - 11: place in GC RAM
- Special Purpose (8-bit, implicit)
    - Program Counter (PC)
    - Memory Address Register (MAR)
    - Memory Buffer Register (MBR)
    - Instruction Register High (IRh)
    - Instruction Register Low (IRl)
    - Stack Pointer
    - Jump register
- Flags
    - Carry Flag
    - Borrow Flag
    - Equal Flag
    - Zero Flag
- Status
    - Error

    - Overflow
    - Instruction Cycle Code (2-bit)
    - Fetch (1-bit): starts at 0 and increments by one once the second byte is fetched (if needed)

Assembler

- Example assembler from Gary Sims: https://github.com/garyexplains/examples/blob/master/vASM.py