EDEMS

Educational DEmonstrative Microprocessor Simulator ## EDEMS registers EDEMS has 16 registers, two of those are 16b, others are 8b. User reachable are 8 registers.

#	———	#	———
R0	H1	R6	L1
R1	H2	R7	L2
R5	A	R4	F
R2	P	R8	C
R3	S	R9	P
R10	OP1	R11	OP2
R12	TMP	R13	u0
R14	uPCH	R15	uPCL

Registers 0-7 are user-addressable registers, 8-15 are microcode-only-addressable registers.

Registers H1 and L1 can be used as 16b register HL1, Registers H2 and L2 can be used as 16b register HL2, Registers PC and SP are only 16b addressable for user.

All registers, except PC(Program Counter) could theoretically be used as general purpose registers, but some bytes of F(flags) is rewrited after almost every ALU operation. SP is used by stack pointer instructions.

A, L1, L2 registers

Those registers can be used as one operand of ALU operations.

OP1, OP2 registers

Are supposed to be used for fetching instruction operands.

TMP register

Main register of most ALU operations.

uO

Indexing register for microinstructions. It can be filled with CuO microinstruction or D2O instruction.

uPCL, uPCH

High and low bits of microprogram counter. Since microprogram addresses are only 11b, 5MSB of uPCH is not used.

Rewriting only one of the registers is not recommended, since you would jump to another part of microprogram. For jump you should use JMP instruction.

NA	NA	NA	NA	NA	uPC[10]	uPC[9]	uPC[8]
uPCH[7]	uPCH[6]	uPCH[5]	uPCH[4]	uPCH[3]	uPCH[2]	uPCH[1]	uPCH[0]

uPC[7]	uPC[6]	uPC[5]	uPC[4]	uPC[3]	uPC[2]	uPC[1]	uPC[0]
uPCL[7]	uPCL[6]	uPCL[5]	uPCL[4]	uPCL[3]	uPCL[2]	uPCL[1]	uPCL[0]

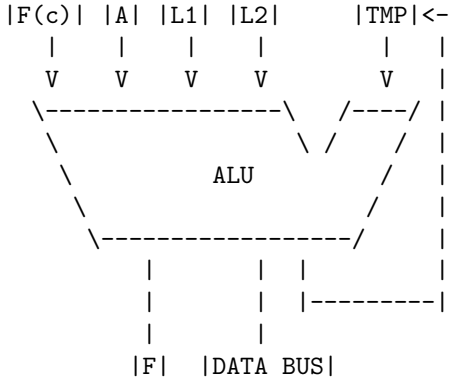
F register

F register contains ALU flags. Those are:

NA	NA	NA	I	V	N	Z	C
F[7]	F[6]	F[5]	F[4]	F[3]	F[2]	F[1]	F[0]

- C - carry
- Z - zero
- N - Negative
- V - Two's complement overflow
- I - Interrupt?

ALUv1



ALU has 5 inputs and 3 outputs. Most of the operations use data bus as its output, but when the output is 16b, TMP register is used for 4 most significant bits.

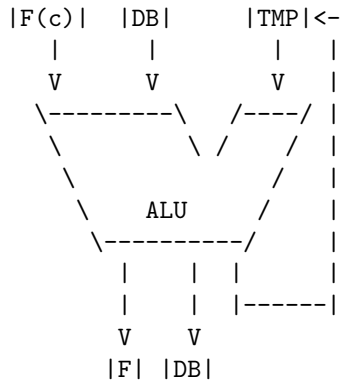
Operations

1. addA - DB = A+TMP
2. subA - DB = TMP + Two'sComplement(A)
3. notA - DB = ~A
4. andA - DB = A && TMP
5. orA - DB = A || TMP
6. mulA - multiply A and TMP
7. rshA - right shift A
8. rrlA - right rotate A
9. rrcA - right rotate through carry A
10. lshA - left shift
11. lrlA - left rotate
12. lrcA - left rotate through carry
13. addL1 - DB = L1+TMP
14. subL1 - DB = TMP + Two'sComplement(L1)
15. notL1 - DB = ~L1
16. andL1 - DB = L1 && TMP

17. orL1 - DB = L1 || TMP
18. mulL1 - multiply L1 and TMP
19. rshL1 - right shift L1
20. rrlL1 - right rotate L1
21. rrcL1 - right rotate through carry L1
22. lshL1 - left shift
23. lrlL1 - left rotate
24. lrcL1 - left rotate through carry
25. addL2 - DB = L2+TMP
26. subL2 - DB = TMP + Two'sComplement(L2)
27. notL2 - DB = ~L2
28. andL2 - DB = L2 && TMP
29. orL2 - DB = L2 || TMP
30. mulL2 - multiply L2 and TMP
31. rshL2 - right shift L2
32. rrlL2 - right rotate L2
33. rrcL2 - right rotate through carry L2
34. lshL2 - left shift
35. lrlL2 - left rotate
36. lrcL2 - left rotate through carry
37. ...

oh my god, this is probably overkill...

ALUv2



ALU has 3 inputs and 3 outputs. Most of the operations use data bus as its output, but when the output is 16b, TMP register is used for 4 most significant bits.

Operations

1. addDB - DB = DB+TMP
2. subDB - DB = TMP + TwosComplement(DB)
3. notDB - DB = ~DB
4. andDB - DB = DB && TMP
5. orrDB - DB = DB || TMP
6. mulDB - DB,MUL = DB*MUL
7. rshDB - right shift DB
8. rrlDB - right rotate DB
9. rrcDB - right rotate through carry DB
10. lshDB - left shift
11. lrlDB - left rotate

12. lrcDB - left rotate through carry
13. equDB - DB = DB == 0

uInstructon set

uInstructions are 2B wide. Opcode usually is 1B with 1 1B operad, but there are some exceptions (JMP is 4b opcode with 11b operand) Every functional block that can have its value (buses, registers) has its own number. They are addressable using that number. u0 register is indexing register. When used in most microinstructions, containing value is used as index instead.

- `CuO + o1` - fill u0 register with corresponding value. u0=IR-o1.
- `ALU + o1` - ALU does operation defined by o1. o1=1=add, o1=2=or, o1=3=not,...
- `2DB + o1` - move value from o1 to DB.
- `2AL + o1` - move value from o1 to AB[0:7].
- `2AH + o1` - move value from o1 to AB[8:15].
- `DB2 + o1` - move value from DB to o1.
- `AB2 + o1` - move value from AB to o1.
- `INC + o1` - increment value in o1.
- `DEC + o1` - decrement value in o1.
- `CP1 + o1` - jump over next microinstruction if value in o1 is 0x00.
- `CF1 + o1` - jump over next microinstruction if value in F(o1) is 0b. uO acts as normal register for this microinstruction.
- `C2D + o1` - move o1 constant to DB.
- `O2D + o1` - move value from uO to DB, o1 is irrelevant.
- `D2O + o1` - move value from DB to uO, o1 is irrelevant.
- `END + o1` - end of microinstruction. Signal for control unit to fetch another instruction.
- `JMP + o1` - write o1 to uPC.
- `HLT + o1` - stop simulator.

brainfuck

used registers:

- HL1 - Data pointer, initial value = `0x0010`
- SP - Output pointer, initial value = `0xFFFF`
- HL2 - Input pointer, initial value = `0x0000`
- OP1 - Parenthesis counter, initial value = `0x00`

Initial memory map

All of memory is filled with `0x00`

```
********** - HL2 - IP - 0x0000
*         *
*         *
********** - HL1 - DP - 0x0010
*         *
*         *
*         *
*         *
*         *
*         *
    ...
*         *
*         *
********** - SP - OP - 0xFFFF
```

Pseudocode for commands

> - 0x3E

Increment the data pointer (to point to the next cell to the right).

```
inc HL1
```

< - 0x3C

Decrement the data pointer (to point to the next cell to the left).

```
dec HL1
```

+ - 0x2B

Increment (increase by one) the byte at the data pointer.

```
inc *HL1
```

- - 0x2D

Decrement (decrease by one) the byte at the data pointer.

```
dec *HL1
```

*** - 0x2A**

Output the byte at the data pointer.

```
*SP = *HL1
dec SP
```

, - 0x2C

Accept one byte of input, storing its value in the byte at the data pointer.

```
*HL1 = *HL2
inc H1L1
```

[- 0x5B If the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching] command.

```
nop
// Special Character
```

] - 0x5D

If the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching [command.

```
counter = 0

while True:
    pointer = pointer-1
    if string[pointer] == "]":
        counter = counter+1
    elif string[pointer] == "[" and counter != 0:
        counter = counter-1
```

```
elif string[pointer] == "[" and counter == 0:
    pointer = pointer+1
    break
```

unknown

Any unknown opcode is skipped.

minimal instruction set

Instructions use two, one ore zero operands. Microcode can operate with "microoperands" that opcodes has coded inside. For example: LDa16 loads 8b value from 16b address. Its definition is `|LDa16 (3b) where| + |addrH| + |addrL|`. every `| |` block means 8bit value. This means that this operation has 8b opcode and two 8b operands. The last 3 bits of opcode defines save location of load action (000b - H1, 111b - L2, 101b - A).

instructions

- `|LD 3b where| + |addrH| + |addrL|` - load from 2B address to 8 bit register.
- `|ST 3b what| + |addrH| + |addrL|` - store from 1B register to 2B address.
- `|ADD 3b where| + |addrH| + |addrL|` - add 1B value from 2B address to 1B register. (result saved in the register)
- `|ADD 2b where| + |addrH| + |addrL|` - add 1B value from 2B address to 2B register. (result saved in the register)
- `|INC2 2b what|` - increment 2B register.
- `|INC 3b what|` - increment 1B register.
- `|DEC2 2b what|` - decrement 2B register.
- `|DEC 3b what|` - decrement 1B register.
- `|TST| + |addrH| + |addrL|` - if value on 2B address is zero, next instruction is skipped.
- `|TSTF 3b which|` - if nth bit of F register is zero, next instruction is skipped.
- `|HLT|` - stop the simulator. Every unknown opcode is HLT.

complete instruction set

instructions

- |LDa16 3b where| + |addrH| + |addrL| - load from 16b address to 8 bit register.
- |LDa8 3b where| + |addr|
- |STa16 3b what|
- |STa8 3b what|
- |ADDa16 3b where|
- |ADDa8 3b where|
- |INC16b 2b what|
- |INC8b 3b what|
- |DEC16b 2b what|
- |DEC8b 3b what|
- |TSTa16|
- |TSTa8|
- |TSTF 3b which|
- ...