“What I cannot create I do not understand”

- Richard Feynman

Simple Decimal Computer

SDC

- Uses *decimal* data. This will make it easier to get started
- 100 memory locations, numbered 00-99
- 10 general purpose registers, numbered R0-R9
- Each memory location holds a signed, 4-digit number

SDC (contd.)

- Control unit has:
  - Instruction Register (IR) – holds the currently executing instruction
  - PC (program counter) – holds the address of the next instruction to execute
  - A flag to indicate whether or not the thing is running
  - **A state machine which you will implement**
  - **Instruction decode logic which you will implement**

Instruction set

- There are 10 unique opcodes, but we have a few more instructions than that in the ISA
- Instructions are 4-digit numbers of the form *±NRMM*
  - Sign is important for only a few operations, is either -1 or 1
  - N is the opcode (0-9)
  - R is a register number (0-9)
  - MM is a memory location (00-99)
  - *Sign is*

Notation

- We indicate the *contents* of register R as `Reg[R]`

- We indicate the *contents* of a memory location MM as `Mem[MM]`

Opcode	*Meaning*	*Implementation*
0	`HALT` execution. (Ignore *R* and *MM*.)	*Running* ← *false*
1	`LD` (Load) Reg[*R*] with the value of memory location *MM*.	`Reg`[*R*] ← `Mem`[*MM*]
2	`ST` (Store) Copy the value of `Reg`[*R*] into memory location *MM*	`Mem`[*MM*] ← `Reg`[*R*]
3	`ADD` contents of location *MM* to `Reg`[*R*]	`Reg`[*R*] += `Mem`[*MM*]
4	`NEG`: Set `Reg`[*R*] to its arithmetic negative (ignore *MM*)	`Reg`[*R*] ← (– `Reg`[*R*])
5	`LDM` (Load immediate): Load `Reg`[*R*] with *MM*	`Reg`[*R*] ← *Sign* * *MM*
6	`ADDM` (Add immediate): Add *MM* to `Reg`[*R*].	`Reg`[*R*] += *Sign* * *MM*
7	`BR` (branch unconditionally): Go to location *MM* (ignore *R*)	`PC` ← *MM*

8	BRC (branch conditional): If *sign* of Reg[*R*] = *Sign*, go to location *MM*	**if** *sign*(Reg[*R*]) = *Sign* **then** PC ← *MM*
90	GETC Read a character and copy its ASCII representation into R0. (A sets R0 = 65, etc.)	Reg[0] ← *Keyboard*
91	OUT Print the character whose ASCII representation is in R0. (A for 65, etc.)	*Print Char*(Reg[0])
92	PUTS Print a string (i.e., the characters) at locations *MM*, *MM+1*, …. Stop (and don't print) when we get to a location that contains 0. (Don't print the zero.)	temp ← *MM* **while** Mem[temp] ≠ 0 *Print* Mem[temp]; temp++
93	DMP Print out the values of the control unit registers (the PC, IR, and the registers)	*Dump Control Unit*
94	MEM Print the values in memory in a 10 × 10 table.	*Dump Memory*
95–99	Ignore instruction	

- For LDM and ADDM, the sign of the instruction indicates the sign of the immediate operand (MM)
- 5123 means “load immediate into R1, where the immediate is 23”
- -5123 menas “load immediate into R1, where the immediate is -23”
- -6201 means R2 <- R2 – 1
- This will be useful since we don’t have a “decrement” instruction

I/O

- Opcode 9 indicates an I/O operation, but there are several variants
- We repurpose the Register number to indicate which variant

Execution

while *Running and* `0 ≤ PC ≤ 99` {

Fetch instruction: `IR ← Mem[PC]; PC++`

Decode instruction: Set Op, R, and MM to `IR[3], IR[2], IR[1:0];`

set Sign to `0, 1` *or* `−1` *depending on* `IR =, >,` or `< 0`.

Get operands, execute instruction, store results:

if *Op* = `0` **then** *Running* ← *False*

else if *Op* = `1` **then** ….

}

You will build an SDC Virtual Machine (VM)

- Lab 5 – Get familiar with SDC, Initialize the control unit, the GPRs, and memory (by loading from a file), some decoding logic
- Lab 6 – Implement a command interpreter (the interface to SDC), implement decode logic and the state machine