Mic-1 microcode notes

This handout elaborates on the 2+ pages of Mic-1 microcode listed in Figure 4-17.

- Each IJVM opcode has a micro-function associated with it. For example, the instruction `SWAP` has micro-instructions `swap1` through `swap6`.
- The ROM address of the first micro-instruction for an opcode is located at the opcode's hex number. For example, the `SWAP` opcode is `0x5F`, so its first micro-instructions, `swap1`, will be located at address `0x5F` in the ROM. That's where they got the funky opcode numbering for IJVM assembly commands.
- Recall that each micro-instruction has a "next address" field. This is the next micro-instruction in the table, unless a goto is specified. For example, the "next address" field of `swap1` is the address of `swap2` (usually the next address in the ROM). The "next address" field of `swap6` is the address of the `Main1` micro-instruction because of the `goto Main1` statement in `swap6`.
- Most micro-functions end with `goto Main1`. This signifies the end of the execution of the current opcode and a return to the main execution loop. The Main1 micro-instruction increments the Program Counter (PC) and fetches the next IJVM opcode.
- The JAM bits (`JMPC`, `JAMN`, `JAMZ`) are complicated, but two cases are the most common. The most common is `000`. This means goto the next micro-instr. Second, JAM bits `100` mean get the next opcode in `MBR`.

The main interpreter loop

The micro-program starts at `Main1` (it may actually start with at `nop1` which just jumps right to `Main1`) and has three parts (all executed in 1 cycle):

```
PC = PC + 1; fetch; goto (MBR)
```

The operations within this micro-instruction:
- increment the program counter
- fetch the next byte in the program, placing the byte in `MBR`
- jump to the opcode in `MBR`

The control and datapath details are:

B bus	Read `PC` onto B
ALU	Disable A input, making it 0; enable B; add with increment (1 on the carry-in bit) resulting in B+1
C bus	Write ALU result into the `PC` register

Memory	Fetch 1 byte from memory at (new) location in `PC` register and store it in lower 8 bits of `MBR`
Next instruction	`JMPC=1`, so jump to micro-instruction at address in the lower 8 bits of `MBR`

Please note that Main1 fetches the byte at PC address in memory before you really know that you need it. In the case of a branch/jump, this byte would be disregarded. There is no inefficiency here, however, as all this happens while the current instruction is being executed.

Encoding a micro-instruction

So, what would the micro-instruction for `Main1` look like?

Field	Bits	Description
`Next addr`	`0 0000 0000`	Set to 0 so when OR'd with MBR, it will equal MBR
`JMPC`	`1`	MPC=MBR, using MBR as next micro-instr addr
`JAMN`	`0`	unused
`JAMZ`	`0`	unused
`SLL8`	`0`	no shift
`SRA1`	`0`	no shift
`F0/F1`	`01`	A or B. But A is 0 (see below), so the result is B
`ENA`	`0`	Use 0 for A
`ENB`	`1`	Enable B input from B bus
`INVA`	`0`	No invert
`INC`	`1`	Increment the PC
`H`	`0`	
`OPC`	`0`	
`TOS`	`0`	
`CPP`	`0`	
`LV`	`0`	
`SP`	`0`	
`PC`	`1`	Set PC register value. PC = ALU result

MDR	0	
MAR	0	
WRITE	0	No mem write
READ	0	No mem read
FETCH	1	Fetch next byte from memory; the next ASM instr
B bus	0001	Place PC value on B bus

So, our final micro-instruction for `Main1` is:

Addr	**Jam**	**ALU**	**C Bus**	**Mem**	**B Bus**
000000000	100	00 01 0101	000000100	001	0001

Paste together the 36 bits: `0000 0000 0100 0001 0101 0000 0010 0001 0001`
Hex: `0 00 15 02 11`