TimBits CPU Language and Pre-Loaded Program

TimBits Assembly Bit Pattern

Each instruction of TimBits language has 8-bits and they each take 8 clock ticks to complete. The instructions have the following pattern (bit 7 is the most significant bit and 0 the least significant bit):

- **Bits 7 to 5: Op-Code**
  - Instructions that use the ALU: arithmetic and conditional branching → b7 is 0
  - Instructions that do not use the ALU: data transfer, input/output and execution → b7 is 1
- **Bit 4: Register/Enabler**
  - Instructions with 2 general purpose registers parameters: add, subtract, conditional branching → bit 4 is 0 when R1,R2 and bit 4 is 1 when R2,R1 (e.g. ADD R1,R2 bit 4 is 0 and it executes R1 = R1 + R2)
  - Instructions with 1 general purpose register parameter: multiply, data transfer, input/output → bit 4 is 0 when R1 and bit 4 is 1 when R2 (e.g. PRT R2 bit 4 is 1 and it displays the number stored in R2 on the 3-bit display)
  - Instructions with 0 general purpose register parameter: stop → bit 4 is 1.
- **Bits 3 to 0: Address/Constant/Fillers**
  - Instructions with an address parameter: conditional branching, data transfer → bits 3 to 0 are a 4-bit RAM address (from 0000 to 1111)
  - Instructions with a constant parameter: multiply → bit 3 is 1 (multiply bit), bits 2 to 0 are a constant ranging from 0 to 7 (decimal, or 001 to 111 in binary)
  - Instructions with no address or constant parameters: add, subtract, input/output, stop → bits 3 to 0 are filler bits, stop is 1111 and all others are 0000.

Category	Instruction	Bit Pattern (bin)
Arithmetic	ADD R1, R2	00000000
	ADD R2, R1	00010000
	SUB R1, R2	01100000
	SUB R2, R1	01110000
	MULT R1, Constant (from 0 to 7)*	00001xxx
	MULT R2, Constant (from 0 to 7)	00011xxx
Conditional Branching	BEQ R1, R2, Address	0010xxxx
	BEQ R2, R1, Address	0011xxxx
	BNQ R1, R2, Address	0100xxxx
	BNQ R2, R1, Address	0101xxxx
Data Transfer	LD R1, Address	1000xxxx
	LD R2, Address	1001xxxx
	STR R1, Address	1010xxxx
	STR R2, Address	1011xxxx
Input / Output	INP R1	11000000
	INP R2	11010000
	PRT R1	11100000
	PRT R2	11110000
Execution	STOP	11111111

Multiplication Circuit:

The multiplication circuit (bonus question) is a separate circuit located inside the ALU. It uses the ALU registers (left, right, a and status). When the MULT command is executed, the 3-bit constant goes through the bit extender (bx) register where it gets expanded to 8 bits before it can be sent to the right register and used by the multiplication circuit.

TimBits Pre-Loaded Program

ROM (ROM Data circuit) currently holds the required multiplication program. Below are the program in TimBits assembly and in C.

In TimBits Assembly:

Program	TimBits Language	Binary	Hex	RAM Address
Load multiplicand into R1	LD R1, 1111	10001111	8f	0000
Load -1 into R2	LD R2, 1101	10011101	9d	0001
Add multiplicand+(-1)	ADD R1, R2	00000000	00	0010
Reassign R1 to multiplicand	STR R1, 1111	10101111	af	0011
Branch if R1 = -1	BEQ R1, R2, 1011	00101011	2b	0100
Load user input to R1	INP R1	11000000	c0	0101
Load current into R2	LD R2, 1110	10011110	9e	0110
Add input+current	ADD R1, R2	00000000	00	0111
Store R1 into current	STR R1, 1110	10101110	ae	1000
Load 0 into R2	LD R2, 0010	10010010	92	1001
Branches to 0000 if input != 0	BNQ R1, R2, 0000	01000000	40	1010
Load current into R1	LD R1, 1110	10001110	8e	1011
Prints final product (current, R1)	PRT R1	11100000	e0	1100
Ends the program execution	STOP	11111111	ff	1101
int current = 0		00000000	00	1110
int multiplicand = 8		00001000	08	1111

In C:

```
int main ( int arc, char **argv ) {
        int current = 0;
        int multiplicand = 2;
        int input = (user input);

        do {
                multiplicand = multiplicand+(-1);
                if (multiplicand == -1) {
                        break;
                }
                current = input + current;
        } while (current != 0);

        print current;
        return 0;
}
```