Your Microprocessor in Action…

Our Microprocessor (for today)

Components:

- Memory: 16 bytes (address: 0 … 15)
- Arithmetic logical unit
- Registers: R0, R1, R2, R3
- Display
- Program counter
- Instruction decoder
- Compiler (not really part of the processor)

Memory

Operations:

- Store a register value into a memory location
- Read a memory location and give it to a register

Simplifications:

- We will allow names for memory locations

Registers

Operations:

- Receive a byte
- Send a byte

Arithmetic Logical Unit (ALU)

Operations:

	COMPUTE		STORE	
A:	R1 + R3	->	R1	Add
B:	R1 + R3 + carry	->	R1	Add with carry
C:	R1 x R3	->	[R0, R1]	Multiply
D:	R1 & R3	->	R1	Bit-wise AND
E:	R1 | R3	->	R1	Bit-wise OR
F:	~R1	->	R1	Bit-wise NOT
G:	-R1	->	R1	2’s Comp Neg
H(x, y):	y	->	Rx	Copy value y to Rx
J(x, y):	Ry	->	Rx	Copy Ry to Rx
T:	R1 – R3		XXXXXXXXX	Compare

Each operation can also update the status register:

SR[zero]:	is the result zero?
SR[negative]:	is the result negative?
SR[carry]:	was there a carry?

Program Memory

- Stores our program

- We will start with C
- For each line of C, our **compiler** will translate into a sequence of “atomic” instructions

Program Counter

Keeps track of which part of the program that we are currently executing

Operations:

- Go to the next line
- Skip up or down multiple lines
- Conditional (on status bit): skip up or down multiple lines

Display

One operation:

- Receive a byte

In response to this operation:

- Convert to written representation
- Write it

Instruction Decoder

Tells everyone what to do….

Sequence:

- Fetch the line of code that is currently indicated by the program counter
- Convert to a sequence of atomic instructions (this is done by our compiler)
- For each operation in order: tell the relevant components what to do
- Repeat

Instruction Decoder

Must determine what is done by each component:

- Memory
- Registers
- Display
- ALU
- Program counter

Program #1

```
uint8_t a;
a = 5;
display(a);
```

Program #2

```
uint8_t a;
a = 5;
a = a + 7;
display(a);
```

Program #3

```
uint8_t a;
uint8_t b;
a = 5;
b = 17;
if (a < b) {
    a = a + b;
}
display(a);
```

Program #4

```
uint8_t a;
uint8_t i;
a = 0;
for(i = 0; i < 4; ++i) {
    a = a + i;
}
display(a);
```

Program #5

```
int8_t a;
int8_t b;
a = 5;
b = a * 100;
display(b);
```

Program #6

```
int16_t a;
int16_t b;
a = 5;
b = a * 100;
display(b);
```

Program #7

```
uint8_t a;
uint8_t i;
a = 0;
for(i = 1; i > 0; i*=2) {
    a = a | i;
    display(a);
}
```

Take-Home Messages

- Many different components
- The components must be coordinated to execute the program properly
- Instructions are translated into a set of control signals for your microprocessor
- Be aware of variable sizes:
  - Small is good for efficiency
  - But the computations that you are performing must fit within these small spaces

Caveats

- Compilation really happens long before execution
- Variable names are handled by the compiler (and disappear before execution)
- Many more registers
  - Variables are stored longer in registers if they are used in consecutive lines (efficiency, but with challenges)
- Many more instructions