Putting Everything Together: An 8 Bit Computer

Nicholas C. Weaver

Why a computer?

- Unified design example
  - Ties together everything
- Familiar Background
  - 61c level knowledge assumed
- A computer requires:
  - Memory
  - Arithmetic Circuits
  - Routing
  - Finite State Machine
- On the file server
  - U:\WVLIB\8bitcpu

What sort of computer?

- 8 bit instruction word
- 8 bit data word
- 4 registers
- Load/Store architecture
  - All operations are on registers
- Separate instruction and data memories
  - Simplifies control logic
  - Increases available memory

Instructions

OP	Format	Meaning
IMM	00 value	Sign extends immediate and places it in R0
BLR0	01 xx Ra Rb	Ra gets PC+1 If Rb = 0, jump to Ra
LD	10 0x Ra Rb	Ra gets Mem[Rb]
SD	10 1x Ra Rb	Mem[Rb] gets Ra
ADD	11 00 Ra Rb	Ra gets Ra + Rb
SUB	11 01 Ra Rb	Ra gets Ra - Rb
NAND	11 10 Ra Rb	Ra gets ~(Ra & Rb)
MOVE	11 11 Ra Rb	Ra gets Rb

If Program Counter is 0xff, processor halts.

A Sample Program (1)

```
# Fibinachi
# R1 = Arg/Return, R2 = Juump Reg
# R3 = Stack Pointer
LI   0xFF     # Load 0xFF into
MOVE R3 R0    # Stack Pointer
LI   6        # Going to calc Fibn
MOVE R1 R0    # of 6
LI   fibn     # Prepare to jump to
MOVE R2 R0    # fibn
LI   0
BLR0 R2 R0    # Jump and link to fibn
LI   0xFF     # Go to end, halts
MOVE R2 R0    # computer.  Answer in
LI   0x0      # R1.
BLR0 R2 R0
fibn:
LI   1
SUB  R3 R0    # Decr stack pointer
SW   R2 R3    # Store Jump location
LI   2
SUB  R3 R0
SW   R1 R3    # Save argument
```

A Sample Program (2)

```
LI   ret1     # Check if N = 2,
MOVE R2 R0    # if so return 1
LI   -2
ADD  R0 R1
BLR0 R2 R0
LI   ret1     # Check if N = 1, if
MOVE R2 R0    # so return 1
LI   -1
ADD  R0 R1
BLR0 R2 R0
LI   2        # Fibn(n-2)
SUB  R1 R0
LI   fibn
MOVE R2 R0
LI   0
BLR0 R2 R0
LI   1        # Store result of
ADD  R0 R3    # Fibn(N-2)
SD   R1 R0
LD   R1 R3    # Reload Argument
LI   1        # Calculate
SUB  R1 R0    # N - 1
```

A Sample Program (3)

```
LI   fibn     # Fibn(N-1)
MOVE R2 R0
LI   0
BLR0 R2 R0
LI   1        # Incr SP
ADD  R3 R0
LD   R2 R3    # Load Fibn(N-2)
ADD  R1 R2    # Fibn(N)
ADD  R3 R0    # Increment SP and
LD   R2 R3    # Load return ptr
ADD  R3 R0
LI   0
BLR0 R2 R0    # Return
ret1:
LI   2
ADD  R3 R0    # Incr SP
LD   R2 R3    # Load return ptr
LI   1
MOVE R1 R0    # Return 1
ADD  R3 R0    # Incr SP
LI   0
BLR0 R2 R0    # Return
```

What Makes up a Computer (1)

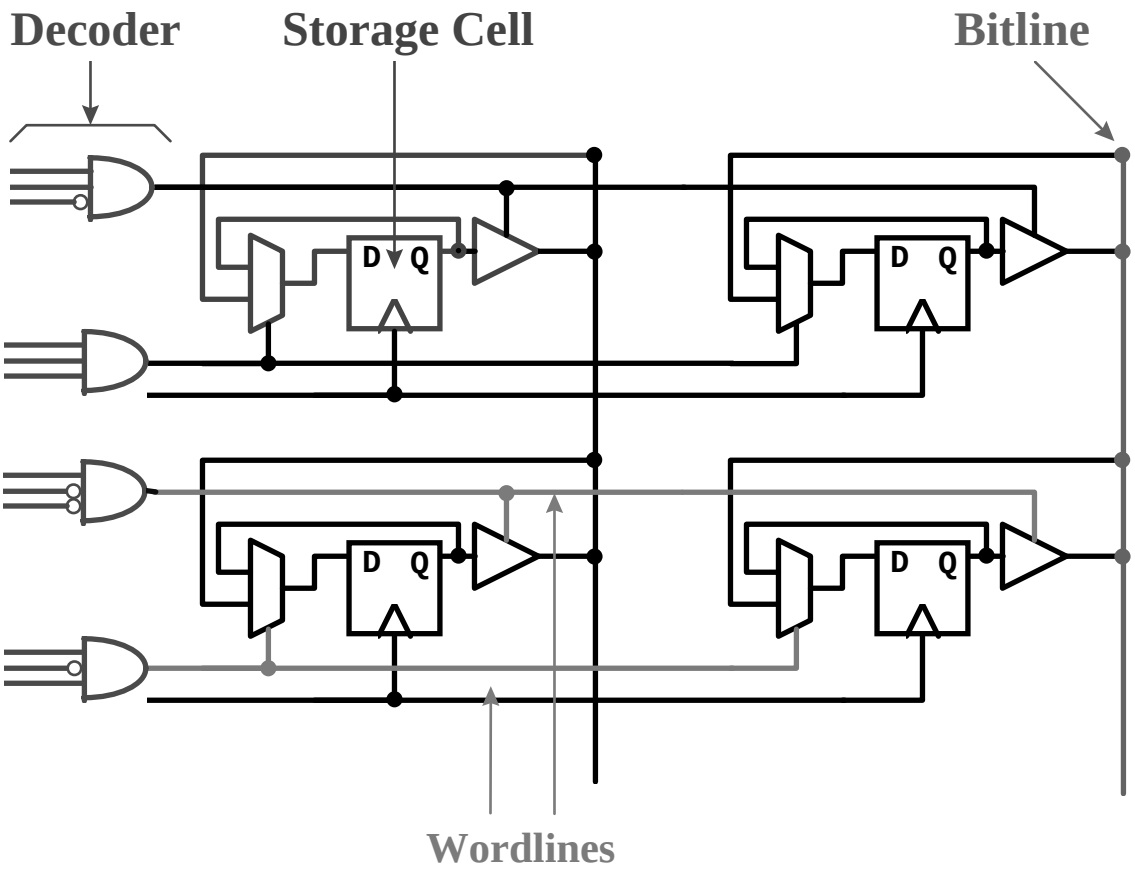
- Data Path:
  - Register File
    - A small, fast memory
  - ALU
  - Memory Bus
    - Also ties together all other components
  - Program Counter and instruction register
    - Holds position in code and current instruction
  - Tied together with muxes and registers

What Makes up a Computer (2)

- Control logic:
  - Finite State Machine
    - Action depends on instruction and possibly register state
    - Less regular, more complicated part of the computer
  - Controls the datapath
    - Switches muxes
    - Latches data in holding registers

Register File

- A small, fast memory



ALU

- Adder/Subtractor
- Multiplier
- Bitwise operations
  - And/or/xor
  - Shifts
- Two arguments input, one output
- Control input
- Selects output with muxes or tri-state buffers

Memory Interface

- Bidirectional Bus to memory
  - Use Tri-state buffers
  - Generally handles all IO
    - Other devices listen for writes to memory addresses
    - Store results into memory and then interrupt the processor

Program Counter

- Has an adder to increment PC by 1
  - Only half adders are needed
- PC can either be loaded from a register or by adding 1 to it
  - Used for branches and jumps
- PC addresses instruction memory
  - Our design uses a separate instruction memory

Finite State Machine & Control logic

- Most of the real complexity is in the FSM design
- Real CPUs may have 100s of states.
  - Split into sub-machines
- Inputs
  - Current instruction
  - Branch?
- Outputs control all datapath lines

Our FSM

- Used a 1 hot finite state machine
- 1 Flip Flop per state
- Looks almost identical to the state transition diagram

So what is a computer

- Datapath
  - Register file
  - ALU
  - Memory
- Tied together with muxes & tristates
- Controlled by a FSM