
An Example Verilog Structural Design: An 8-bit MIPS Processor

Peter M. Kogge

Using design “mips.v” by Neil Weste and David Harris

in “CMOS VLSI Design, 3rd Ed”

with code as found at <http://www.cmosvlsi.com/>

under the “Spice and Verilog code” link

The ISA (see pp. 39-49 of Weste & Harris)

- ❑ 32 bit instructions as in your text book
- ❑ Only eight general purpose registers \$0 to \$7
 - Each register only 8 bits
 - \$0 is hardwired to 00000000
- ❑ PC is also only 8 bits wide
- ❑ All data accesses are only 8 bits, not 32 bits
- ❑ Only opcodes:
 - R format: ADD, SUB, AND, OR, SLT,
 - I format: ADDI, BEQ, LB, SB
 - J format: J
- ❑ *NOTE: Code as presented does not implement ADDI!*

The Implementation

- ❑ Based on *Multicycle* implementation of Chap. 5 of your text
- ❑ Memory is 256 words of 8-bits each

Orange: inter module connections

Red: signals from “top”

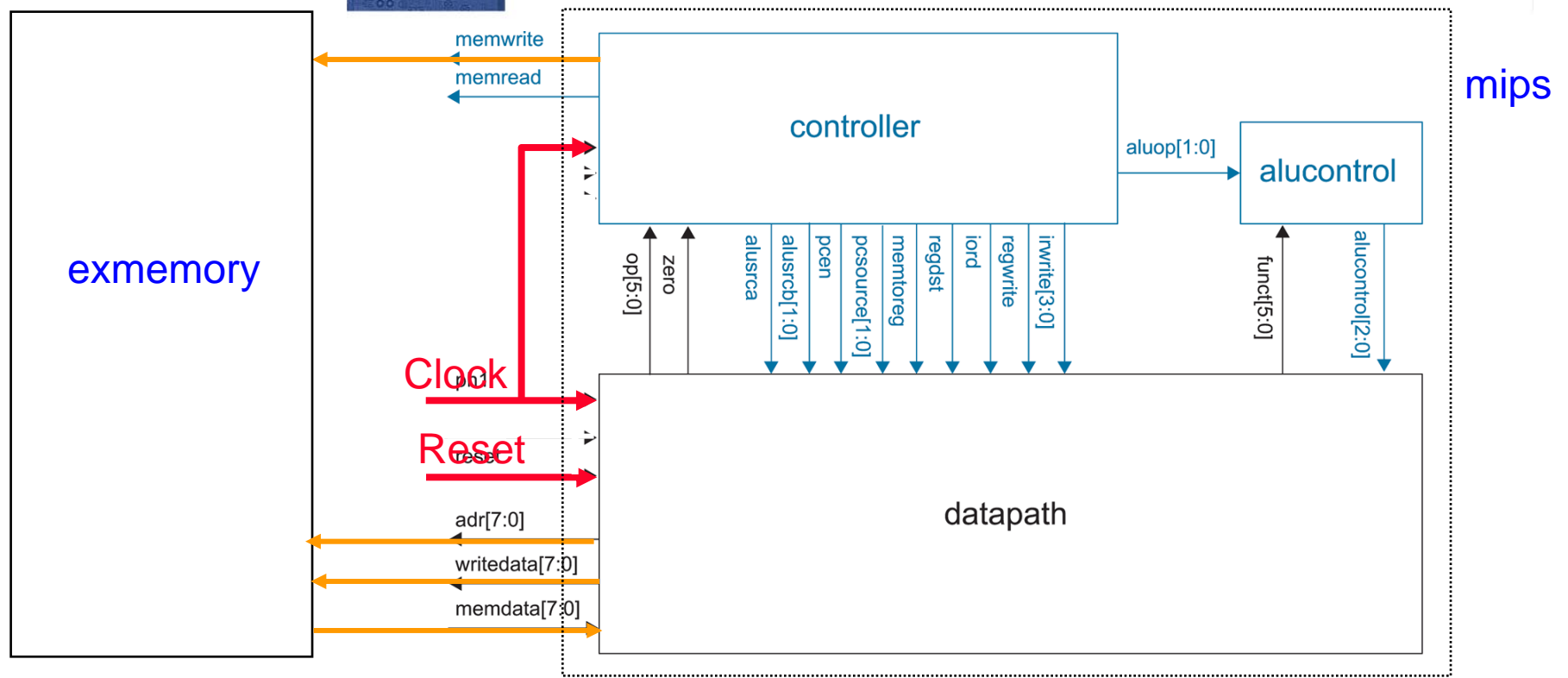


FIG 1.56 Top-level MIPS block diagram

Modules

- ❑ **top**: top level testbench code to configure & test processor
 - **exmemory**: 256x8-bit single ported memory
 - **mips**: the processor itself
 - **controller**: “behavioral” multi-cycle state machine that generates control signals
 - **alucontrol**: “behavioral” decodes aluop & funct fields into ALU signals
 - **datapath**: “structural” Datapath design
 - **flop**: 8-bit flip-flop latch always latched on rising clock edge
 - » Used for all internal staging registers *mdr*, *areg*, *wrd*, *res*
 - **flopen**: 8-bit flip-flop latch with an enable
 - » Used for four instruction register pieces *ir0*, ... *ir3*
 - **flopenr**: 8-bit flip-flop latch with an enable and a reset to zero
 - » Used for *pcreg*
 - **mux2**: 2 input 8-bit wide multiplexer
 - **mux4**: 4 input 8-bit wide multiplexer
 - **alu**: alu description
 - **regfile**: 3-port register file description
 - **zerodetect**: logic to detect all zeros in an 8-bit path

Memory

- ❑ From outside memory is 256 words of 8-bits each
 - Separate *writedata* and *memdata* ports
- ❑ Internally 64 words of 32-bits each
 - Upper 6 bits of *adr* used to select which word
 - Lower 2 bits of *adr* used to select which byte
- ❑ At initialization, loaded from a file named “memfile.dat”
 - Whose format is as a “.csv” like file
 - Where each line in file is contents of a 32-bit word
 - And each word expressed as 8 hexadecimal digits
 - With the 1st word going into word[0], the next into word[1], etc
 - You do not need to load the whole memory
- ❑ During operation, it is always “reading” to *memdata*
- ❑ Write operation occurs “at” rising edge of clock
 - *adr* and *writedata* presented at same time as *memwrite* goes to 1

Top module (similar to “testbench”)

- ❑ Instantiates the *mips* core and the *exmemory*, and interconnects them
- ❑ Starts with raising *reset* to 1 for 22 time units, then dropping it
- ❑ Also generates a clock of 10 time unit period
- ❑ Also includes a load program specific termination test:
 - If the program ever writes to location 5
 - And the data is a “7”, then success
 - Else failure
- ❑ Writing from *writedata* into the memory occurs on the rising edge of the clock

Controller module (Behavioral)

❑ States

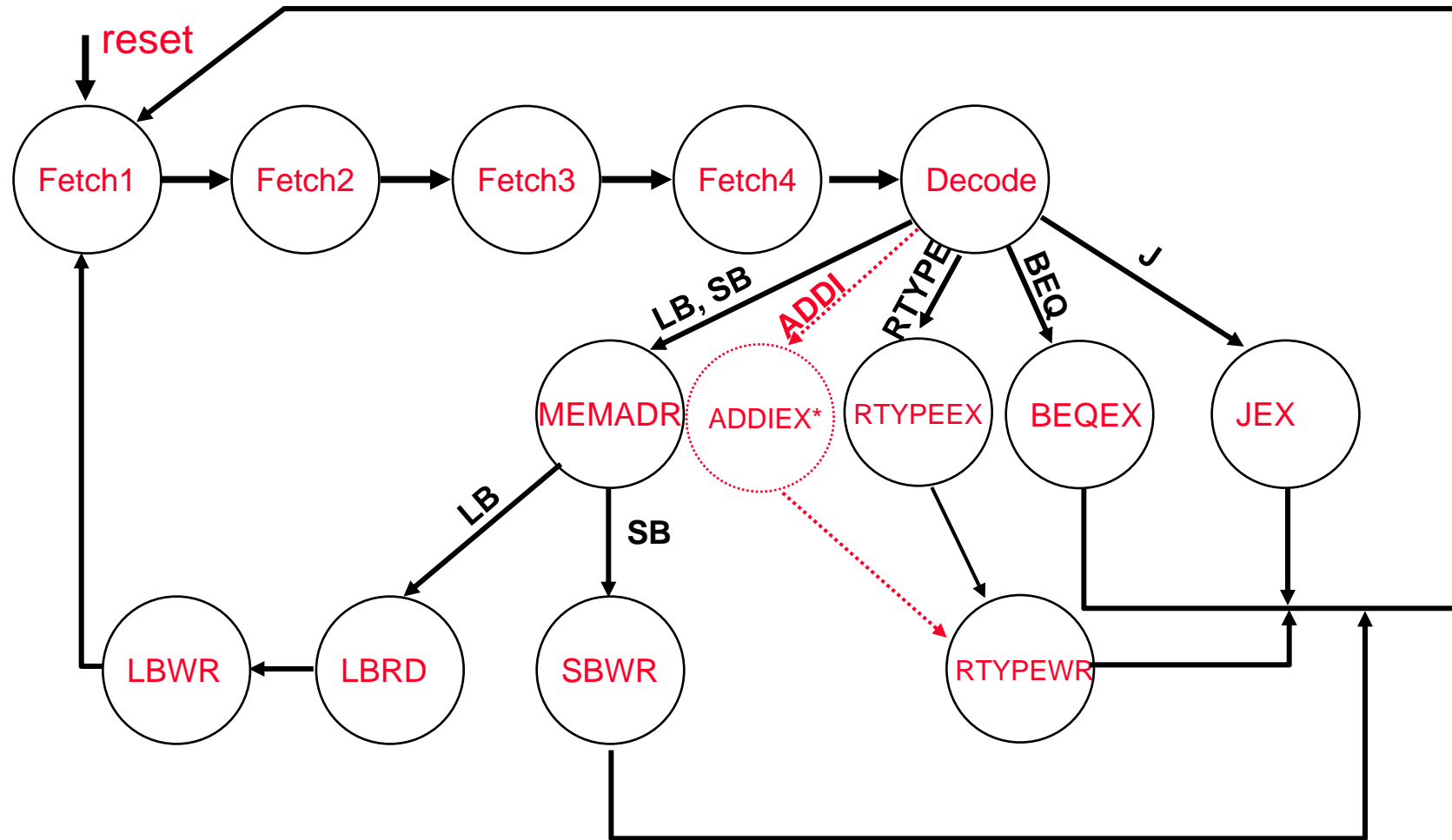
- FETCH1, FETCH2, FETC3, FETCH4: 4 states to read 32b instruction
- DECODE: decode just fetched instruction
- MEMADR: computes a memory address
- RTYPEEX: execute R-type opcode
- RTYPEWR: write result back at end of R-type opcode into reg file
- LBRD: read data from memory into core
- LBWR: write data just read from memory into reg file
- SBWR: write data to memory
- BEQEX, JEX: execute states for BEQ or J opcodes
- *ADDIEX: new state for ADDI implementation*

❑ Reset changes state to *FETCH1* state

❑ Internal state changes on rising edge of clock

❑ Control signals assume their values starting at rising clock

State Diagram



* added for ADDI implementation

Instruction Cycle Table

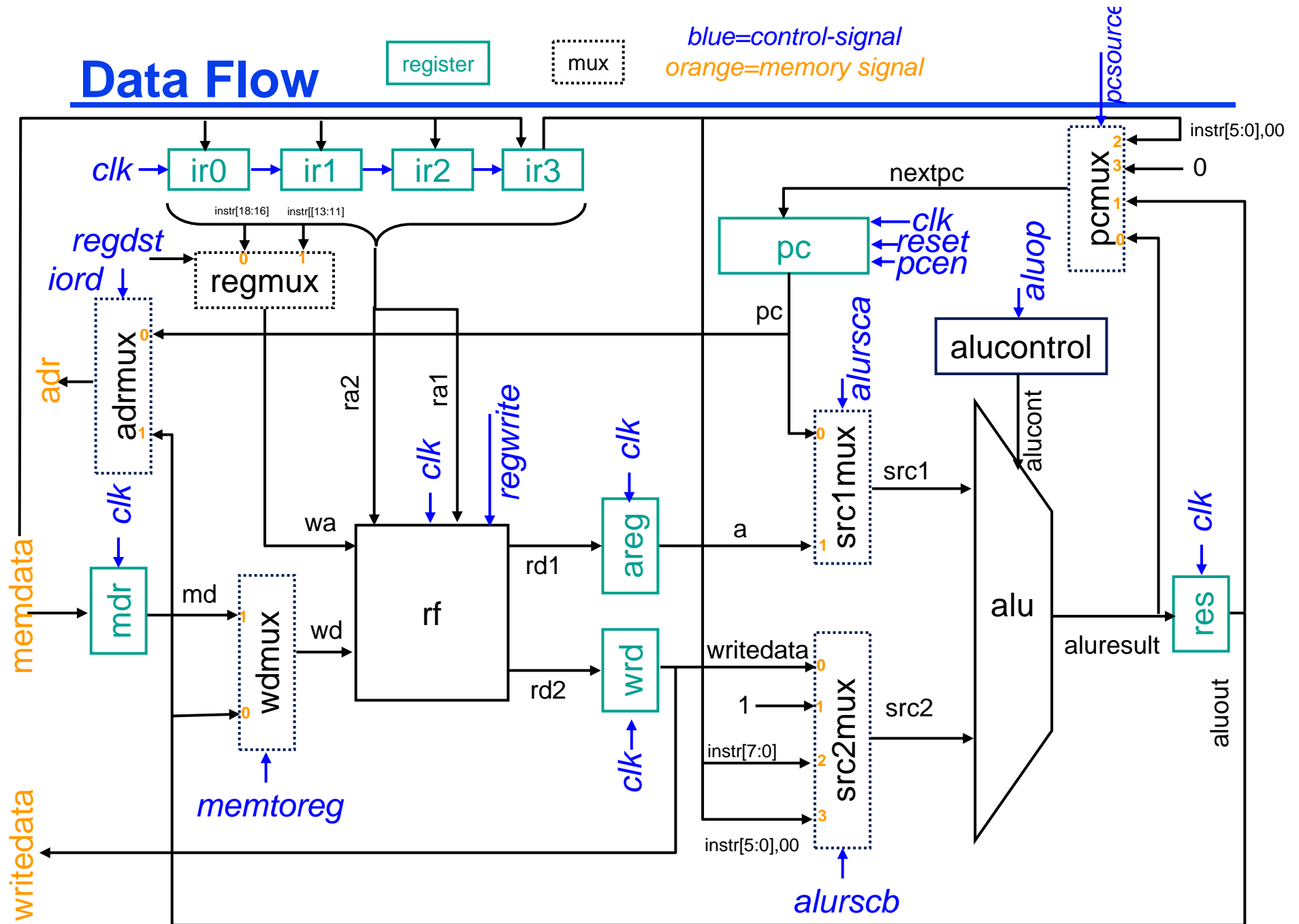
Opcode	# Cycles	Cycles (Starting with DECODE)
ADD	7	DECODE, RTYPEX,RTYPEWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
ADDI	7	DECODE, ADDIEX,RTYPEWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
AND	7	DECODE, RTYPEX,RTYPEWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
BEQ	6	DECODE, BEQEX,IFETCH1,IFETCH2,IFETCH3,IFETCH4
J	6	DECODE, BEQEX,IFETCH1,IFETCH2,IFETCH3,IFETCH4
LB	8	DECODE, MEMADR,LBRD,LBWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
OR	7	DECODE, RTYPEX,RTYPEWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
SB	7	DECODE, MEMADR,SBWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
SLT	7	DECODE, RTYPEX,RTYPEWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4
SUB	7	DECODE, RTYPEX,RTYPEWR, IFETCH1,IFETCH2,IFETCH3,IFETCH4

Data Flow

register

mux

blue=control-signal
orange=memory signal



Register File module

- ❑ 2 read, 1 write port
- ❑ Always reading on read ports
 - I.e. change the register address on *ra1* or *ra2* and *rd1*, *rd2* change immediately
- ❑ Writing occurs at rising edge of clock
 - if *regwrite* signal is active

Datapath module

- ❑ Instruction register implemented as 4 8-bit latches
 - ir0, ... ir3
 - Loaded sequentially during IFETCH
- ❑ *pcreg*:
 - Reset to zero on a *reset* high
 - Loaded from *pcmux*
 - ALU used to increment pc
- ❑ Includes internal staging latches (store on rising edge)
 - *areg*: capture output of read port 1 of reg file
 - *wrd*: capture output of read port 2 of reg file
 - *res*: capture output of alu
 - *mdr*: capture read data output from memory