8-BIT MICROPROCESSOR DESIGN AND LAYOUT

SUSAN ERALY

JOSH GROSSBERG

MARCIN TKACZYK

OBJECTIVE

This project involved the architectural and layout design of an 8 bit RISC microprocessor capable of performing simple arithmetic operations and data transfer instructions. The final design is to fit in a 40 pin pad frame and not take up area greater than 2.5mm X 2.5mm. This project was the culmination of a semester long course in VLSI system design and involved utilizing all the skills and knowledge gained from the course.

INSTRUCTION SET

The instruction set consists of thirteen simple instructions as described in Table 1.

TABLE 1

Mnemonic	opcode, operand, operand	Description
NOP	0000	No operation
LOAD Register, PC	0001, Register address	Loads the contents of the program counter into the register specified
LOAD PC, Register	0010, Register address	Loads the contents of the register specified into the program counter
LOAD Register1, Register2	0011 Register address, Register address	Loads the contents of Register2 into that of Register1. Contents of Register 2 remain unchanged.

BRZ Address	0100, Program memory address	Branches to the program memory address specified if the zero flag is set
BRE Address	0101, Program memory address	Branches to the program memory address specified if the equal flag is set
BRL Address	0110, Program memory address	Branches to the program memory address specified if the less than flag is set
BROV Address	0111, Program memory address	Branches to the program memory address specified if the overflow flag is set
HLT	1000	Halts program execution.
STORE Address, Register	1001, Register address, Data memory address	Stores the contents of the register into the memory address specified
LOADM Address, Register	1010, Register address, Data memory address	Loads the register specified with the contents of the memory address
NOP	1011	No operation
CMPE Register1, Register2	1100, Register address, Register address	Compares the contents of register1 and register2 and sets the equal flag if they are equal
ADD Register1, Register2	1101, Register address, Register address	Adds the contents of register 1 to that of register2 and stores the result in register1. The overflow and the zero flag are set to

		reflect the status of the addition
SHFTL Register	1101, Register address	Shifts the content of the specified register left and stores it back into it.
CMPL Register1, Register2	1111, Register address, Register address	Compares the contents of register1 and register2 and sets the less than flag if the content of Register1 is less than that of Register2.

The microprocessor, since it is based on RISC philosophy, has only two addressing modes. Data can be transferred from a 1Kbyte data memory into any of the 8 registers in the Register File or from register to register

The instruction word is hence 17 bits long to account for the worst case scenario where the 10 bit program/data memory address along with the 3 bit long register file address and the 4 bit opcode have to be accommodated.

DATA PATH

The design of the data path was the first step in the entire design process. Figure1. gives the 8-bit data path designed.

The main blocks of the data path are the PC, the Register File, the IRQ, the ALU and the special purpose registers PSR, WIR and WRES.

The program status register or the PSR is a 4-bit register that holds the zero, equal, less than and overflow flags. The WIR register hold the address of the data memory that needs to be accessed. WRES serves the double purpose of being the dummy register while transferring data between the register file and the memory. Data that is read from the data memory or has to be written into the memory is stored in the WRES.

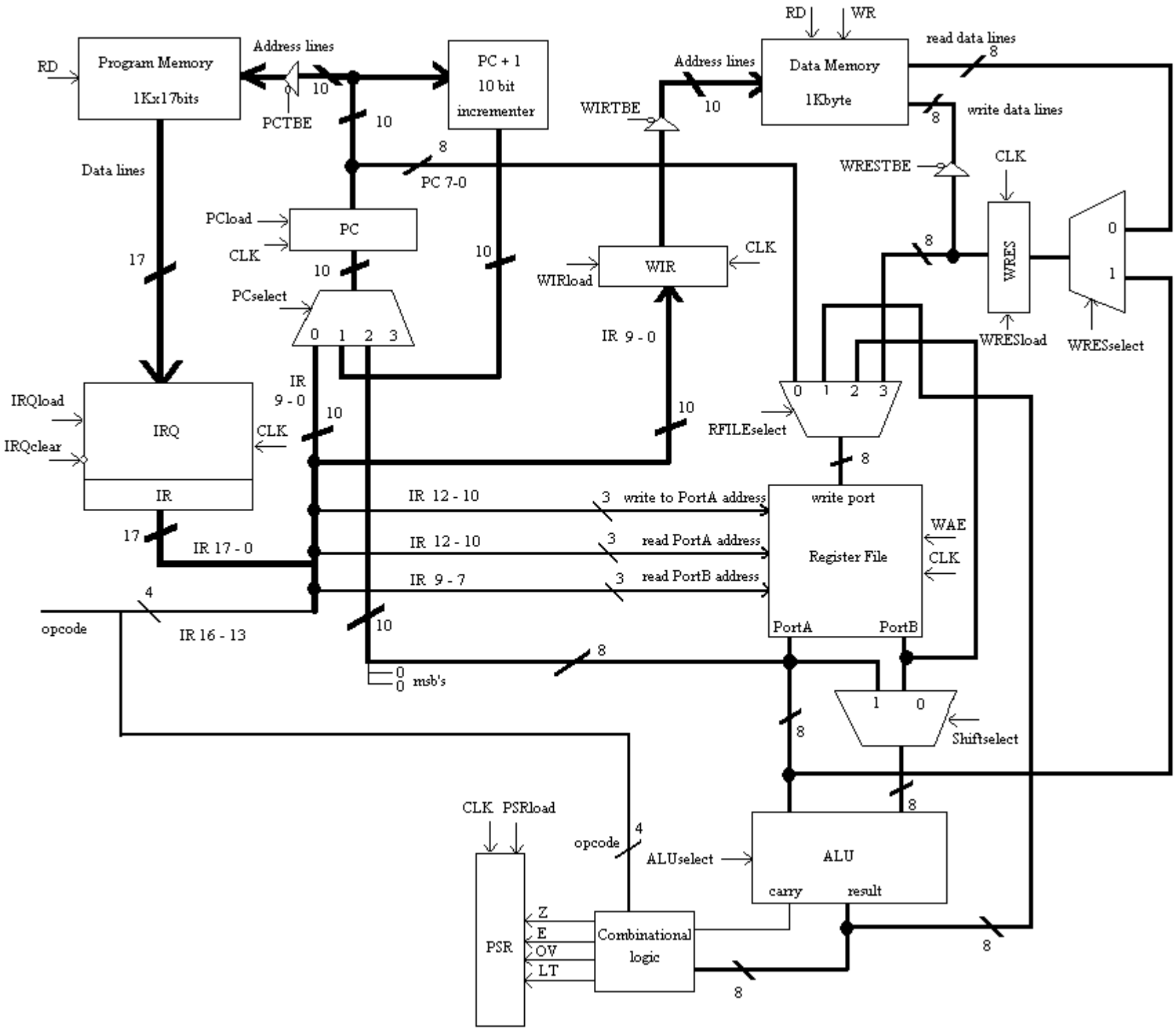

RD
Program Memory
1Kx17bits
Address lines
10
PCTBE
PC + 1
10 bit
incrementer
Data lines
17
8
PC 7-0
PCload
PC
CLK
10
PCselect
0 1 2 3
IR
9 - 0
IRQload
IRQclear
IRQ
CLK
IR
17
IR 17 - 0
4
opcode
IR 16 - 13
10
0
0
msb's
WIRTBE
WIR
WIRload
CLK
IR 9 - 0
RD
WR
Address lines
10
Data Memory
1Kbyte
read data lines
8
write data lines
WRESTBE
CLK
WRES
WRESload
WRESselect
0
1
0 1 2 3
RFILEselect
8
IR 12 - 10
3
write to PortA address
IR 12 - 10
3
read PortA address
IR 9 - 7
3
read PortB address
write port
Register File
WAE
CLK
PortA
PortB
1 0
Shiftselect
CLK PSRload
opcode
4
ALUselect
ALU
carry
result
PSR
Z
E
OV
LT
Combinational
logic

FIG1

REGISTER FILE

The register file is an array of eight 8-bit registers. The register file has two read ports and one write port. The data at the write port of the register is written into the address specified by WA2 WA1 WA0 when the write signal to Port A, WAE, is asserted. The contents of the register specified by the addresses WA2 WA1 WA0 and RB2 RB1 RB0 are output at PortA and PortB respectively. There are no specific read enable signals.

ALU

The ALU is a simple 8-bit adder subtractor that shifts left by adding the data to itself and hence the multiplexer at the second input of the ALU. The ALU gives an 8-bit result along with the carry, the ninth bit. The signal “aluselect” dictates whether an addition or subtraction operation is to be performed, depending on the opcode. Comparison to check for less than and equal conditions are performed with subtraction.

Multiplexers are provided at the input of the PC, the register file and WRES as these registers accept values from various sources depending on the instruction being executed.

The register file accepts the contents of the PC in the Load Register, PC address. It accepts the result of the ALU after the addition or the shift operation. It accepts the contents of PortB for Load Register1, Register2 instruction or the contents of the WRES register in the Load Register, Address instruction.

The PC accepts the result of the PC incrementer ordinarily. But if the jump condition is satisfied the PC has to load the last 10 bits of the instruction word. If the instruction is Load PC, Reg the value that needs to be loaded into the PC is the output of PortA of the Register File. Hence the multiplexer at the input of the PC register. The last input of the mux is left unused.

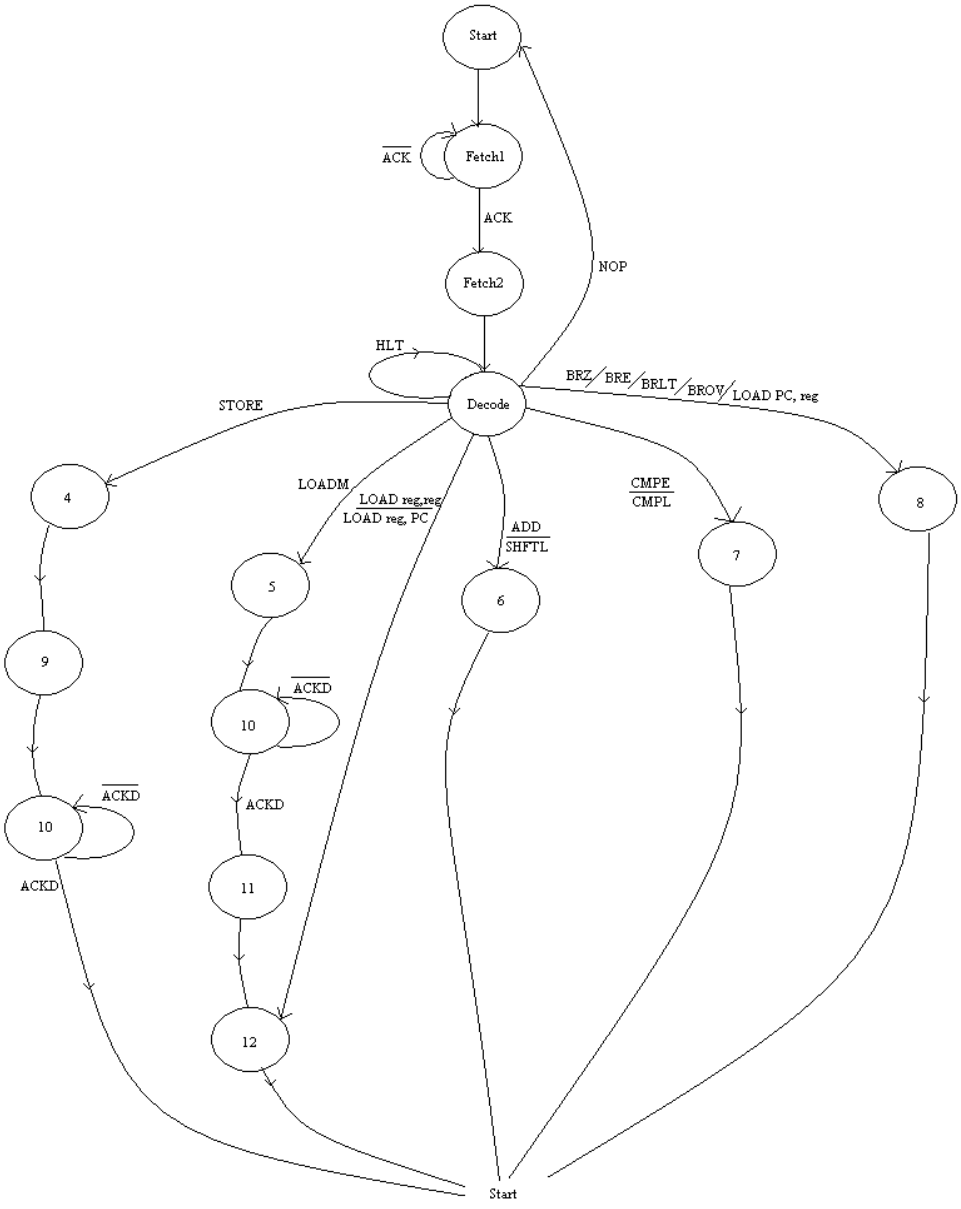
To make the data path and control unit logic simpler the last 10 bits, which could specify a jump address or a data memory address in certain cases, is hardwired to the mux at the input of the PC and the WIR register. Bits 12-10 specify the address of the register and in other cases the bits 9-7 specify the address of the second register and are hence hardwired to the read and write addresses of the register file.

The opcode is given by bits 16-13 of the instruction register.

Tristate buffers are provided between the registers that hold the address to the program and data memory and the address lines of the program and data memory and also between the register that holds the data to be written into the data memory and the write data lines. In order to make sure that a valid address is available at the address lines before the read or write signals are sent out, the read and write signals are delayed a half clock cycle.

CONTROL UNIT

The operation of the processor is dictated by the control unit. The control unit sends out signals that enable the tri state buffers, load the various registers, and determine the select inputs to the various mux's. These signals are in turn controlled by the first four bits of the instruction word along with a 4 bit, 14 state finite state machine. Attached are the state diagram and the state table for the finite state machine.



State diagram

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ack	ack'	ackd(op10)	ackd(op9)	ackd'
CurrentState0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					
1																	2	1			
2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3					
3	0	12	8	12	8	8	8	8	3	4	5	0	7	6	6	7					
4										9											
5											10										
6														0	0						
7													0			0					
8			0		0	0	0	0													
9										10											
10																			11		11	10
11											12										
12											0										

STATE TABLE

The next state equations were derived from this table and implemented using two 4 X 16 decoders that decode the opcode and the previous state respectively. The majority of the signals were controlled either entirely by the opcode or entirely by the state of the machine. Below is the table for the opcode dependant control signals.

Opcode / Control Signal	0 NOP	1 LD Reg, PC	2 LD PC, Reg	3 LD Reg1, Reg2	4 BRZ	5 BRE	6 BRL	7 BROV	8 HLT	9 STORE	10 LOADM	11 NOP	12 CMPE	13 ADD	14 SHFTL	15 CMPL
PCselect1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
PCselect0	1	1	0	1	0	0	0	0	X	1	1	1	1	1	1	1
Rfileselect1	X	0	1	1	X	X	X	X	X	X	1	X	X	0	0	X
Rfileselect0	X	0	1	0	X	X	X	X	X	X	1	X	X	1	1	X
ALU select	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	1
Shift select	X	X	X	X	X	X	X	X	X	X	X	X	0	0	1	0
WRESselect	X	X	X	X	X	X	X	X	X	1	0	X	X	X	X	X

One can observe from this table that the opcode dependant signals are all mux select inputs. Below is the table for the state dependant signals.

State / Load	0	1	2	3	4	5	6	7	8	9	10	11	12
WRESload	0	0	0	0	1	0	0	0	0	0	0	1	0
WIRload	0	0	0	1	0	0	0	0	0	0	0	0	0
WAE	0	0	0	0	0	0	1	0	0	0	0	0	1
PSRload	0	0	0	0	0	0	1	1	0	0	0	0	0
IRQload	0	0	1	0	0	0	0	0	0	0	0	0	0
PCTBE	1	1	0	0	0	0	0	0	0	0	0	0	0
WIRTBE	0	0	0	0	1	1	0	0	0	1	1	0	0

These signals are all enabling and loading signals. Hence the select signals need only be correct when the registers whose inputs they are selecting are being loaded. This

simplified the design significantly by allowing for the number of states in the finite state machine to be greatly reduced.

Certain signals were more complicated and were dependant on both state and opcode. Below are the equations for the signals PCload, IRQclear viz. active low and WRESTBE.

$$PCload = State2 + State8\ (op4 \times Z + op5 \times E + op6 \times L + op7 \times OV + op\ 2)$$

$$\overline{IRQclear} = State8\ (op4 \times Z + op5 \times E + op6 \times L + op7 \times OV)$$

$$WRESTBE = State9 + State10 \times op9$$

PCload and IRQclear were complicated because at state 8 they are each dependant on certain flags being sent from the PSR. The tristate buffer to WRES is enabled at state 10 only during the store instruction.

The read and write signals were complicated because they were required to be one half clock cycle later than the enables of the respective tri state buffers that produced valid addresses at the outputs. The equations for these signals are shown below.

$$RDDM = State5 \times \overline{CLK} + Op10 \times State10$$

$$WRDM = State9 \times \overline{CLK} + Op9 \times State10$$

$$RDPM = State0 \times \overline{CLK} + State1$$

The IRQclear signal is sent through a delay flip-flop as the IRQ needs to be cleared only after the finite state machine has exited State 8.

CONCLUSION

Attached are simulations of the NOP, BRE, ADD, LOADM, STORE, CMPE instructions. Also attached is a simulation of the working of the register file. The layout images of the microprocessor, the IRQ, the control unit, the Register file and the ALU are also included.