University of Pennsylvania
Department of Electrical and System Engineering
Computer Organization

ESE680-002, Spring 2007 Assignment 3: Instructions Monday, January 29

Due: Wednesay, February 7, 12:00PM

Everyone should do problems 1–5. You should use a drawing program for datapaths (where appropriate).

1. Consider a simple, sequential, non-branching, programmable datapath with a single one-bit output computational unit. For this problem consider two possible functional units: a 3-input NAND and a 3-input LUT. Now, let us consider implementing a 16-input parity function (XOR16) on each of these programmable datapaths.

   - Datapath should be universal; that is it should still be possible to compute any function using this datapath.

   (a) Draw your datapth for each case.

   (b) Define the primitive instruction (pinst) for each of the units (what bits are included, what do they do, how many of each).

   (c) How many instruction bits are required to specify the computation for each instruction in the two cases?

   (d) How many instruction cycles will it take to implement the parity function in each of these cases? (show assembly to support)

   (e) How many total operation instruction bits in memory are required to describe this operation in each case?

2. Consider the branching datapath shown at the end of the assignment (basically unchnaged from the previous assignment). Concretely, consider the datapath width to be 16. Implement code to compute the parity of a 16b word in two cases:

   (a) Using no branching operations, attempting to minimize compute cycles.

   (b) Using branching operations, attempting to minimize instructions.

   For each case:

   - provide assembly code
   - report cycles to perform the computation
   - report instructions required to describe the computation

3. Continue considering the datapath and 16b word parity operation from Problem 2. Add an instruction to the datapath that computes this parity.

   - How many gates does this addition require?
     (a) instruction decoding
     (b) logic
     (c) datapath multiplexing
     - count 2-input gates; for concreteness, you may assume the ALU bitslice shown in class as starting point (though it may not quite support the existing operations assumed)
   - Assuming each gate takes $2500\lambda^2$ and each instruction bit takes $1200\lambda^2$, report the area difference (savings or addition) resulting from adding this operation to the datapath compared to the implementation in Problem 2 with the fewest instructions.

4. Continuing to consider the 16b parity and the base datapath, explore the impact on cycles and instruction compactness of changing the datapath to support 2 or 1 register operations.

   (a) base case is the existing datapath. *i.e.* 2 source registers and one destination register in each instruction: rdst = rsrc1 op rsrc2 [so, no new coding here, just fillin your results from Problem 3 in the summary table requested below.]

   (b) 2 register instruction: rdst = rsrc op rdst (allow overwrite in single instruction with operation)

   (c) 1 operand/instruction: accum = accum op rsrc (see new 1-operand ALU operation table below)

aluop	operation
ADD	accum← accum+rsrc
INV	accum← ∼(accum)
SUB	accum← accum-rsrc
XOR	accum← accum∧rsrc
OR	accum← accum|rsrc
INCR	accum← accum+1
AND	accum← accum&rsrc
BNZ	if (src1!=0) pc←branch_addr
SRA	accum← accum>>1; accum[31]=accum[31]
SRL	accum← accum>>1; accum[31]=0
SLA	accum← accum<<1
SLL	accum← accum<<1
STO	rsrc← accum
ZERO	accum← 0
LD	accum← rsrc
DONE	stop execution

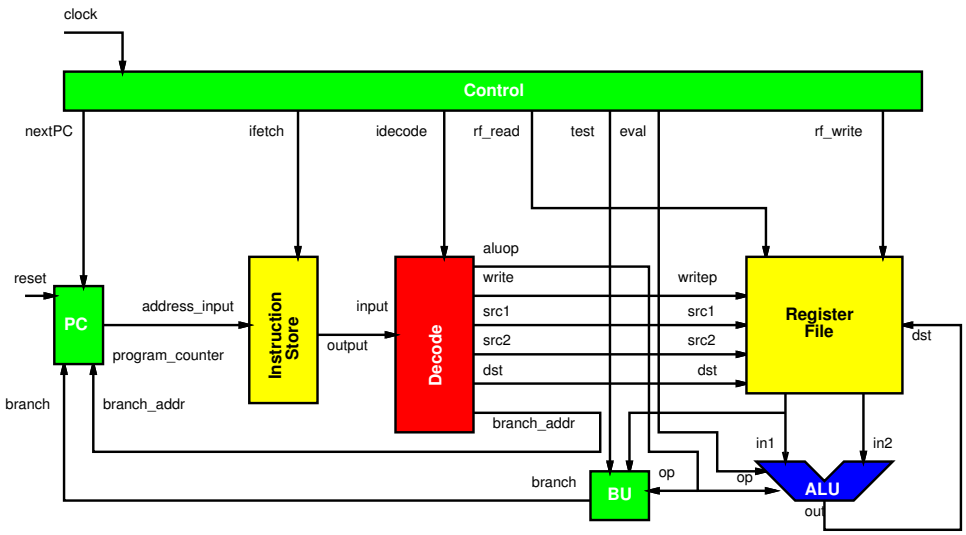
   - Consider the branching case.
   - Sketch enough assembly code to justify your answer.

   Complete the following table based on your results.

Architecture	Total Cycles for parity	pinst width	total bits for parity
3 register			
2 register			
1 register			

5. Let's say you have an old design which is 70% instruction memory, and you've picked an optimized datapath and instruction encoding scheme to reduce the instruction memory size by 35% while keeping other things the same. Assume, for simplicity, technology is continuously improving such that you get a reduction in feature size by a factor of 2 every three years. How many months of technology scaling give the same size reduction as your improved design?

Simple Branching Processor Datapath



The basic processor organization is as shown above. Non-branching instructions are of the form:

bits	13:10	9	8:6	5:3	2:0
field	op	w	src1	src2	dst

- op – operation to be performed (typically by ALU)
- w – write back ALU output to register file? (1=yes, 0=no)
- src1 – address of first ALU operand in register file
- src2 – address of second ALU operand in register file
- dst – address in register file into which the result should be sotred

For branch operations, the branch_addr is the low 6 bits of the instruction; that is, it is in the same place we would have placed src2 and dst in a normal, ALU operation.

bits	13:10	9	8:6	5:0
field	BNZ	0	src1	branch_addr

Generally, on each cycle the processor performs:

```
op,w,src1,src2,dst = instruction_store[pc]
...,branch_addr = instruction_store[pc]
in1=register_file[src1]
in2=register_file[src2]
if (w==1)
  register_file[dst]←(in1 op in2)
if ((op==BNZ) && (in1!=0))
  pc←branch_addr
else
  pc←pc+1
```

A special "done" operation indicates the computation is done and the program counter should stop incrementing. A reset signal tells the program counter to set its value to zero and begin computation.

The following ops are defined:

aluop	encoding	operation
ADD	0x00	dst← src1+src2
INV	0x01	dst← ∼(src1)
SUB	0x02	dst← src1-src2
XOR	0x03	dst← src1∧src2
OR	0x04	dst← src1|src2
INCR	0x05	dst← src1+1
AND	0x08	dst← src1&src2
BNZ	0x0A	if (src1!=0) pc←branch_addr
SRA	0x0B	dst← src1>>1; dst[31]=src1[31]
SRL	0x0C	dst← src1>>1; dst[31]=0
SLA	0x0D	dst← src1<<1
SLL	0x0E	dst← src1<<1
DONE	0x0F	stop execution