CSL216
Pipelined ARM Assembly Simulator

Ankit Mahlawat(2016CSJ0006)
Ashish Dagar(2016CSJ0010)

May 3, 2018

Contents

Chapter 1

Objective

In this assignment we will enhance the instruction set simulator developed in Assignments 3 and 4, to also incorporate pipelining. We will try to avoid data and control hazards and ensure functional correctness. We will print out statistics about the clock cycle counts and average Instructions Per Cycle (IPC) for the input program. Our program will be capable of displaying, after each instruction, the various instructions that are currently residing in the pipeline, along with the stage that they are in.

Chapter 2

function and Structures

Structure are implemented of type IFIDStruct,IDEXStruct, EXMEMStruct, MEMWBStruct and WBENDStruct

IFIDStruct which is for process going between Instruction fetch and instruction decode.
It contains elements- instr ,pcPlus1 both of integer type.

IDEXStruct which is for process going between Instruction decode and execution stage.
It contains elements-instr,pcPlus1, readRegA, readRegB,offset all of integer type.

EXMEMStruct which is for process going between Execution and memory stage.
It contains elements- instr,branchTarget,aluResult,readRegB all of integer type.

MEMWBStruct which is for process going between memory and write back stage.
It contains elements- instr,writeData both of integer type.

Structure stateStruct tells about current state of pipeline.
It contains following elements with their type specified as-
int pc;
int instrMem[NUMMEMORY];
char dataMem[NUMMEMORY];
int reg[NUMREGS];

int numMemory;
IFIDType IFID;
IDEXType IDEX;
EXMEMType EXMEM;
MEMWBType MEMWB;
int cycles

printState is a function for printing state of pipeline.
It has following fields of integer type-
field0 - It gives destination.
field1 - It gives source 1.
field2- It gives source 2 which is either register or immediate.
opcode-It gives unique id for every instruction.

printInstruction is a funtion which prints which type of instruction it is.

read_word(char *mem, unsigned int address)

void write_word(char *mem, unsigned int address, unsigned int data);

Chapter 3

Branch prediction

A 2-bit branch prediction variable h0 is declared then in if stage branch is predicted and pesudocode for that is

```
if (instruction in IF stage is branch )
then
if (h0 has a state 2 or 3  )
then
newState pc <- target branch pc
```

now prediction result is handled in execution stage where branch to be taken gets clear

```
if (branch instruction) // branch prediction
    then
            if (branch to be taken is true )
            then
            if (state of h2 is not taken) {
                    flush instruction
                    and change pc
            else if(h0 is not equal 3) increase h0 by 1


      else if(branch instruction){
     if (h2 state is taken)
     flush instruction behind branch
     update pc
    else if(h0 not equal 0) decrease h0 by 1
```

Chapter 4

Instruction Fetch

In this stage irrespective of type of instruction every instruction is fetched from instruction memory.
In case of unconditional branch , branch is taken in this stage.
Normally the program counter is increased by 4 in every clock cycle except in case of branching where it is increased by the required offset.

Instruction is passed from pc state to IF/ID stage.
Pc is increased by 1

```
newState pc increase by 1
if (unconditional branch instruction)
        newState pc is given target branch pc
else if (unconditional branch with linking)
        newState pc is given target branch pc
        newState reg 14 is given target branch pc;
```

Chapter 5

Instruction Decode

In this stage instruction is passed to execution stage.
Registers and immediate are read irrespective of instruction type.

```
read register one
read register two
read offset of branch

\\stalls for LDR is checked and executed here
if (ldr instuction in execution stage and
    rd of ldr equals register to be read in later instruction
    and later instuction is alu)
then stall i.e pc—
ifid instuction equals no op instruction
```

Chapter 6

Execution

In this stage instructions depending on their type are executed accordingly.
If it is add instruction then the wo operands are added.
If it is of type load then address is calculated.
I have used two variable integer tempregA and tempregB . for execution stage computation.

```
      // start forwarding
if (EX/MEM stage Register Rd = ID/EX stage RegisterRn1
and EX/MEM instruction is a writeback instruction)
then
 tempRegA is given this state EX/MEM aluResult;
else if (MEM/WB stage Register Rd = ID/EX stage Register Rn1
and MEM/WB instr is a writeback instruction)

 tempRegA is given this state MEM/WB writeData
else if(ID/EX instruction is alu instuction)

tempRegA is given this state Rn1


if (EX/MEM stage Register Rd = ID/EX stage Register Rm2
and EX/MEM has writeback operation and perand two is not
immediate in ID/EX instruction)

tempRegB is given this state EX/MEM stage aluResult;
else if(MEM/WB.RegisterRd = ID/EX.RegisterRm2 and MEM/WB
has writeback operation and perand two is not immediate
in ID/EX instruction )
```

```
tempRegB is given this state MEM/WB writeData
else if(ID/EX has alu instuction)

if (Rm2 is not imm.)

tempRegB is given register Rm2 value
else

tempRegB is given immediate value from instruction


//end forwarding


if (ID/EXinstr is AND)

        newState.EXMEM.aluResult  is given (tempRegA & tempRegB) ;

else if (ID/EX instr is ADD)

        newState.EXMEM.aluResult  is given tempRegA + tempRegB;

else if (ID/EX instr is ORR)

   newState.EXMEM.aluResult  is given (tempRegA | tempRegB);

else if (ID/EX instr is SUB)

        newState.EXMEM.aluResult  is given (tempRegA - tempRegB);

else if (ID/EX instr is MUL)

        newState.EXMEM.aluResult  is given (tempRegA * tempRegB);

else if (ID/EX instr is CMP)

  if (tempRegA = tempRegB)
                Z = 1;
  else
                Z = 0;
```

```
    if (tempRegA is les than tempRegB)
                N is given 1;
        else
                N is given 0;
else if (ID/EX instr is MOV)

        newState EXMEM aluResult  is given  tempRegB;
else if (ID/EX instr is MVN)

        newState EX/MEM aluResult  is given  ~(tempRegB);
```

Chapter 7

Memory Access

In this stage instructions which do not require memory access such as add , subtract,multiply,orr etc are passed as it is from this stage without any change.
In load instruction the value stored at specified address is read for writing it in required register in further stage.
In store instruction the value which was read from the source register in previous stages is now written at the appropriate address in memory.

```
//ldr str forwarding
if (MEM/WB Register Rd = EX/MEM Register Rn1
and EX/MEM.instr = LDR and MEM/WB.instr has WB operation)

  ldrA is given this state MEM/WB writeData;
else if(EX/MEM.instr = LDR)

  ldrA is given value of register Rn1
else if(MEM/WB Register Rd = EX/MEM Register Rn1
and EX/MEM.instr = STR and MEM/WB.instr has WB operation)

  ldrA is given this state MEMWB writeData;
else if( EX/MEM.instr = STR)

  ldrA is given value of register Rn1

if (MEM/WB Register Rd = EX/MEM Register Rd
and EX/MEM.instr = STR and MEM/WB.instr has WB operation)
```

```
  ldrB is given this state MEMWB writeData;
else if (EX/MEM.instr = STR )

  ldrB is given value of register Rd

 //end forwarding

if (EX/MEM.instr = LDR && register Rn1 is less than 14 )

  newState.MEMWB.writeData is given
  read_word(state.dataMem,ldrA + immediate)
  //read_word function reads data from memory
else if(EX/MEM.instr = LDR && register Rn1 is = 14 ))

  newState.MEMWB.writeData is given
  read_word(state.dataMem,immediate);

else if(EX/MEM.instr = LDR)

  newState.MEMWB.writeData is given immediate
else if (EX/MEM.instr = STR &&  register Rn1 is less than 14 )

  write_word(newState dataMem , (ldrA + immediate) , ldrB )
  //write_word function writes data to memory
else if (EX/MEM.instr = STR)

  write_word(newState dataMem , immediate , ldrB);

else if(EX/MEM.instr is alu instr with WB operation)

  newState MEMWB writeData is given this state EXMEM aluResult;
```

Chapter 8

Write Back

In this stage the instructions which do not require write in registers such as store are passed as it is from this stage without any change.
In load instruction the value which was read in previous stage from memory is written in appropriate register.
In instructions of type add ,subtract etc the result obtained is written in destination register.

```
if (instruction in writeback stage is a write instruction)
  then
  perform write operation in register given in instruction
```

Chapter 9

pesudocode for statistics of Program

int variable MEMcycle,EXcycle,tcycles,fcycles are used in the process. bool variable change is used to check weather branch prediction has failed or passed.

```
if (instruction memory stage is LDR)
then
  MEMcycle is given ldr
else if (instruction memory stage is STR)

  MEMcycle is given str
else
  MEMcycle is given 1

if(opcode(state.IDEX.instr) = AND)

  EXcycle is given and1
else if (instruction execution stage is SUB)

  EXcycle is given sub
else if (instruction execution stage is ADD)

  EXcycle is given add
else if (instruction execution stage is MUL)

  EXcycle is given mul
else if (instruction execution stage is CMP)
```

```
  EXcycle is given cmp
else if (instruction execution stage is ORR)

  EXcycle is given orr
else if (instruction execution stage is MVN)

  EXcycle is given mnv
else if (instruction execution stage is BNE or  BEQ) && change)

  EXcycle is given bne
else if (instruction execution stage is BLE ||  BGT ||
BGE)  && change)
then
  EXcycle is given bge
else

  EXcycle is given 1

if (instruction in IF stage is B)

  tcycles is given b
else if (instruction in IF stage is BL)

  tcycles is given bl
else
  tcycles is given 1

if (EXcycle is greater than MEMcycle)
     if (EXcycle>tcycles) fcycles is given EXcycle ;
     else fcycles is given tcycles;
else
    if (MEMcycle is greater than or equal to tcycles)
    fcycles is given MEMcycle;
    else fcycles is given tcycles;
```

now fcycles will decide the number of cycles for which stall is needed.

THE END