**17/06/20**

**פרויקט מספר 2 – מבנה מחשבים מגיש דוד מוסייב**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\*\*PROJECT 2 Computer arangemnt\*

\*\*\*author: David Musaev\*\*\*\*\*\*\*\*\*

\*\*\*Ruppin accademic center\*\*\*\*\*\*/

/\*MIPS Pipeline Simulator\*/

/\*1.The simulator will handle each command and show handling of each buffer in the pipeline by showing the command

2.The simulator will handle hazard by stalls , fowarding, branc resolution

3.the simulator will count CPI\*/

/\*defines\*/

#define LINESIZE 100

/\*flags of hazards care\*/

#define FW\_FLAG 1 //fowarding flag hazzard care

#define BRNCHRES\_FLAG 1 //branch resolution hazzard care

/\*structs\*/

/\*command definition\*/

typedef struct COMMAND

{

char cmd\_type[10];//1.lw 2.addi 3.addi 4.bne

char rs[10];//register Source $1-$10

char rt[10];//register Target $1-$10

char immidiate[10];// immidiate value

char memory\_address[10];//memory address

}command;

typedef struct stages

{

char bufferNum[5][LINESIZE];

}stages;

/\*function decleration\*/

void pipelineMain(char \*traceFile,int flags);

void moveFoward(stages \*thisCyclePipeLineStages, char \*tmpLineHandel);

void printStatusPipe(stages \*thisCyclePipeLineStages, int CYCLE, char \*filename);

/\*main\*/

void main()

{

char trace1[] = "trace1.txt";

char trace2[] = "trace2.txt";

int flagHazzard = 0;

printf("\nTrace1.txt \n\n");

pipelineMain(trace1,flagHazzard);

printf("\nTrace2.txt \n\n");

pipelineMain(trace2,flagHazzard);

flagHazzard = 1;

printf("\nTrace1.txt \n\n");

pipelineMain(trace1, flagHazzard);

printf("\nTrace2.txt \n\n");

pipelineMain(trace2, flagHazzard);

flagHazzard = 10;

printf("\nTrace1.txt \n\n");

pipelineMain(trace1, flagHazzard);

printf("\nTrace2.txt \n\n");

pipelineMain(trace2, flagHazzard);

flagHazzard = 11;

printf("\nTrace1.txt \n\n");

pipelineMain(trace1, flagHazzard);

printf("\nTrace2.txt \n\n");

pipelineMain(trace2, flagHazzard);

}

/\*functions\*/

/\*1.pipeline- this function will show in which stage\*/

void pipelineMain(char \*traceFile, int flags)

{

/\*variables decliration\*/

char tmpLineHandel[LINESIZE];//line handler

int numOfLines = 0;//lines of the fille

int i = 0,j=0;

char commandName[10],tmpRegName[10];

int Hazzardflag = 0;

float CPcounter = 0, CPI = 0;

int CYCLES = 0; ;

char newName[20];

/\*files openning\*/

FILE \*trace = fopen(traceFile, "r+");//open for read and write and update

snprintf(newName, sizeof(newName), "OUT %d %s", flags,traceFile);//unite the line into one string

FILE \*traceOUT = fopen(newName, "w+");

/\*get the line until the EOF of trace\*/

while(!feof(trace))//count the number of the lines in the file

{

fgets(tmpLineHandel, LINESIZE, trace);

numOfLines++;

}

rewind(trace);

/\*commands defintion and memory allocation\*/

command \*thisCommand[5];//5 stages of pipeline

for (int i = 0; i < 5; i++)

{

thisCommand[i] = (command\*)calloc(1, sizeof(command));//current command

}

i = 0;//reset i

stages \*thisCyclePipeLineStages = (stages\*)calloc(1, sizeof(stages));//declaration of this cycle pipeline stages

/\*reset the the cycles\*/

strcpy(thisCyclePipeLineStages->bufferNum[0], "NULL");

strcpy(thisCyclePipeLineStages->bufferNum[1], "NULL");

strcpy(thisCyclePipeLineStages->bufferNum[2], "NULL");

strcpy(thisCyclePipeLineStages->bufferNum[3], "NULL");

strcpy(thisCyclePipeLineStages->bufferNum[4], "NULL");

char tmpLineHandel1[LINESIZE];

strcpy(tmpLineHandel , "stall");

/\*scan the lines\*/

for (j = 0; j <= numOfLines; j++)

{

//command format: #address #commandtype #registerTarget #registerSource #immidiate R type commands

fscanf(trace,"%s %s %s %s %s",&thisCommand[i]->memory\_address,&thisCommand[i]->cmd\_type,&thisCommand[i]->rt,thisCommand[i]->rs, &thisCommand[i]->immidiate);

snprintf( tmpLineHandel1,sizeof(tmpLineHandel1),"%s %s %s %s %s",(thisCommand[i]->memory\_address) , (thisCommand[i]->cmd\_type) , (thisCommand[i]->rt) , (thisCommand[i]->rs) , (thisCommand[i]->immidiate) );//unite the line into one string

strcpy(commandName ,thisCommand[i]->cmd\_type);//handle the command name only for cheking if its needed an new arrangment

/\*addi or subi command or add\*/

if ((strcmp(commandName, "addi")==0) || (strcmp(commandName, "subi") == 0) || strcmp(commandName, "add") == 0)

{

CPcounter += 4;

}

/\*lw or sw command arangment\*/

if (strcmp(commandName, "lw") == 0 || strcmp(commandName, "sw") == 0)

{

if (strcmp(commandName, "lw") == 0)CPcounter += 5;

if (strcmp(commandName, "sw") == 0)CPcounter += 4;

/\*swap rs and immidiate\*/

strcpy(tmpRegName, thisCommand[i]->rs);

strcpy(thisCommand[i]->rs, thisCommand[i]->immidiate);

strcpy(thisCommand[i]->immidiate, tmpRegName);

}

/\*bneq command arangment\*/

if (strcmp(commandName, "bneq") == 0)

{

CPcounter += 3;

/\*swap rs and rt\*/

strcpy(tmpRegName, thisCommand[i]->rt);

strcpy(thisCommand[i]->rt, thisCommand[i]->rs);

strcpy(thisCommand[i]->rs, tmpRegName);

}

/\*if there is a hazzard and its not the first command\*/

if (Hazzardflag == 1 && (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0) || (strcmp(thisCommand[i + 3]->rt, thisCommand[i]->rs) == 0) || (strcmp(thisCommand[i + 4]->rt, thisCommand[i]->rs) == 0)))//worst case if the \*Rs is a \*Rt and still in the pipeline

{

/\*if flag fowarding =0 and flag branch flag=0\*/

if (flags=0)

{

for (int k = 1; k < 4; k++)

{

\*thisCommand[i + k]->cmd\_type = "stall";

\*thisCommand[i + k]->memory\_address = " ";

\*thisCommand[i + k]->rt = " ";

\*thisCommand[i + k]->rs = " ";

\*thisCommand[i + k]->immidiate = " ";

moveFoward(thisCyclePipeLineStages, tmpLineHandel);//send stall

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

}

}

/\*if flag fowarding =0 and flag branch flag=1\*/

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0) || (strcmp(thisCommand[i + 3]->rt, thisCommand[i]->rs) == 0))

if (flags=10)

{

for (int k = 1; k < 4; k++)

{

/\*branch hazard control & predict\*/

if (strcmp(thisCommand[i + 3]->cmd\_type, "bneq")==0 && atoi(thisCommand[i+2]->memory\_address) != 4 + atoi(thisCommand[i+3]->memory\_address))//if there is a branch in EX stage ALU out is =0

{

/\*stall two commands before the branch\*/

\*thisCommand[i + 1]->cmd\_type = "stall";

\*thisCommand[i + 1]->memory\_address = " ";

\*thisCommand[i + 1]->rt = " ";

\*thisCommand[i + 1]->rs = " ";

\*thisCommand[i + 1]->immidiate = " ";

\*thisCommand[i + 2]->cmd\_type = "stall";

\*thisCommand[i + 2]->memory\_address = " ";

\*thisCommand[i + 2]->rt = " ";

\*thisCommand[i + 2]->rs = " ";

\*thisCommand[i + 2]->immidiate = " ";

strcpy(thisCyclePipeLineStages->bufferNum[3], thisCyclePipeLineStages->bufferNum[2]);

strcpy(thisCyclePipeLineStages->bufferNum[2], "stall");

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

moveFoward(thisCyclePipeLineStages, tmpLineHandel);//send stall.

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

break;

}

/\*stalls\*/

\*thisCommand[i + k]->cmd\_type = "stall";

\*thisCommand[i + k]->memory\_address = " ";

\*thisCommand[i + k]->rt = " ";

\*thisCommand[i + k]->rs = " ";

\*thisCommand[i + k]->immidiate = " ";

moveFoward(thisCyclePipeLineStages, tmpLineHandel);//send stall

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

}

}

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0) || (strcmp(thisCommand[i + 3]->rt,thisCommand[i]->rs) == 0))

/\*if flag fowarding =1 and flag brach flag=0\*/

if (flags=1)

{

/\*Fowarding in MEM and EX stage\*/

for (int k = 1; k < 4; k++)

{

\*thisCommand[i + k]->cmd\_type = "stall";

\*thisCommand[i + k]->memory\_address = " ";

\*thisCommand[i + k]->rt = " ";

\*thisCommand[i + k]->rs = " ";

\*thisCommand[i + k]->immidiate = " ";

moveFoward(thisCyclePipeLineStages, tmpLineHandel);//send stall

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0))

if ((strcmp(commandName, "lw") != 0 || strcmp(commandName, "sw") != 0) && (k ==2 ))//if EXE fowarding

{

strcpy(thisCommand[i + k]->rs, thisCommand[i + k]->rt);

break;

}

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0))

if (k == 3)//if MEM fowarding

{

strcpy(thisCommand[i + k]->rs, thisCommand[i + k]->rt);

}

}

}

/\*if flag fowarding =1 and flag brach flag=1\*/

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0) || (strcmp(thisCommand[i + 3]->rt, thisCommand[i]->rs) == 0))

if (flags=11)

{

for (int k = 1; k < 4; k++)

{

\*thisCommand[i + k]->cmd\_type = "stall";

\*thisCommand[i + k]->memory\_address = " ";

\*thisCommand[i + k]->rt = " ";

\*thisCommand[i + k]->rs = " ";

\*thisCommand[i + k]->immidiate = " ";

moveFoward(thisCyclePipeLineStages, tmpLineHandel);//send stall

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

/\*branch hazard control & predict\*/

if (strcmp(thisCommand[i + 3]->cmd\_type, "bneq") == 0 && atoi(thisCommand[i + 2]->memory\_address) != 4 + atoi(thisCommand[i + 3]->memory\_address))//if there is a branch in EX stage ALU out is =0

{

/\*stall two commands before the branch\*/

\*thisCommand[i + 1]->cmd\_type = "stall";

\*thisCommand[i + 1]->memory\_address = " ";

\*thisCommand[i + 1]->rt = " ";

\*thisCommand[i + 1]->rs = " ";

\*thisCommand[i + 1]->immidiate = " ";

\*thisCommand[i + 2]->cmd\_type = "stall";

\*thisCommand[i + 2]->memory\_address = " ";

\*thisCommand[i + 2]->rt = " ";

\*thisCommand[i + 2]->rs = " ";

\*thisCommand[i + 2]->immidiate = " ";

strcpy(thisCyclePipeLineStages->bufferNum[3], thisCyclePipeLineStages->bufferNum[2]);

strcpy(thisCyclePipeLineStages->bufferNum[2], "stall");

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

moveFoward(thisCyclePipeLineStages, tmpLineHandel);//send stall.

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

break;

}

/\*Fowarding in MEM and EX stage\*/

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0))

if ((strcmp(commandName, "lw") != 0 || strcmp(commandName, "sw") != 0) && (k == 2))//if EXE fowarding

{

strcpy(thisCommand[i + k]->rs, thisCommand[i + k]->rt);

break;

}

if (strcmp(thisCommand[i + 1]->rt, thisCommand[i]->rs) == 0 || (strcmp(thisCommand[i + 2]->rt, thisCommand[i]->rs) == 0))

if (k == 3)//if MEM fowarding

{

strcpy(thisCommand[i + k]->rs, thisCommand[i + k]->rt);

}

}

}

}

/\*proceed foward in the pipeline\*/

moveFoward(thisCyclePipeLineStages, tmpLineHandel1);

printStatusPipe(thisCyclePipeLineStages, CYCLES, newName);

CYCLES++;

for (int k = 4; k >0; k--)//command history in the pipeline

{

strcpy(thisCommand[k ]->memory\_address, thisCommand[k-1]->memory\_address);

strcpy(thisCommand[k ]->cmd\_type, thisCommand[k-1]->cmd\_type);

strcpy(thisCommand[k ]->rt, thisCommand[k-1]->rt);

strcpy(thisCommand[k ]->rs, thisCommand[k-1]->rs);

strcpy(thisCommand[k ]->immidiate, thisCommand[k-1]->immidiate);

}

Hazzardflag = 1;

}

CPI = CPcounter / CYCLES;

printf("\nThe CPI of %s is %f", traceFile, CPI);

fprintf(traceOUT, "The CPI of %s is %f\n", traceFile, CPI);

fclose(traceOUT);

}

void moveFoward(stages \*thisCyclePipeLineStages,char \*tmpLineHandel)

{

strcpy(thisCyclePipeLineStages->bufferNum[4], thisCyclePipeLineStages->bufferNum[3]);

strcpy(thisCyclePipeLineStages->bufferNum[3], thisCyclePipeLineStages->bufferNum[2]);

strcpy(thisCyclePipeLineStages->bufferNum[2], thisCyclePipeLineStages->bufferNum[1]);

strcpy(thisCyclePipeLineStages->bufferNum[1], thisCyclePipeLineStages->bufferNum[0]);

strcpy(thisCyclePipeLineStages->bufferNum[0], tmpLineHandel);

}

void printStatusPipe(stages \*thisCyclePipeLineStages, int CYCLE, char \*filename)

{

FILE \*OUTPUTfile = fopen(filename, "a+");

printf("\nCycle number =%d", CYCLE);

fprintf(OUTPUTfile,"\nCycle number =%d", CYCLE);

printf("\nthe fetch instruction is:%s", thisCyclePipeLineStages->bufferNum[0]);

fprintf(OUTPUTfile, "\nthe fetch instruction is:%s", thisCyclePipeLineStages->bufferNum[0]);

printf("\nthe decode instruction is:%s", thisCyclePipeLineStages->bufferNum[1]);

fprintf(OUTPUTfile, "\nthe decode instruction is:%s", thisCyclePipeLineStages->bufferNum[1]);

printf("\nthe execute instruction is:%s", thisCyclePipeLineStages->bufferNum[2]);

fprintf(OUTPUTfile, "\nthe execute instruction is:%s", thisCyclePipeLineStages->bufferNum[2]);

printf("\nthe memory instruction is:%s", thisCyclePipeLineStages->bufferNum[3]);

fprintf(OUTPUTfile, "\nthe memory instruction is:%s", thisCyclePipeLineStages->bufferNum[3]);

printf("\nthe write back instruction is:%s \n", thisCyclePipeLineStages->bufferNum[4]);

fprintf(OUTPUTfile, "\nthe write back instruction is:%s \n", thisCyclePipeLineStages->bufferNum[4]);

fclose(OUTPUTfile);

}