Single cycle

Homework 2
모바일시스템공학과
32143153 이대훈

1. Project Introduction

Mips green shit을 사용하여 돌아가는 single cycle을 만든다.
Assembly code에서 사용하는 register와 instruction을 사용하여 execute한다.
결과 값은 Register number 2에 저장한다.

2. Motivation

Low level language를 High level language를 통해 실행하도록 하였다. 이를 통해 Low level language에 대한 이해와 실행을 위한 High level language 사이의 연결과정을 알 수 있다.

3. Concepts used in Single cycle

Mips green shit의 실행문 사용
Instruction을 각각 bit masking을 사용한 의미 확인
htonl을 사용한 리틀 엔디안에서 빅 엔디안으로 변경된 값 저장
Data path를 통한 각각의 함수 사용(Fetch, Decode, Execute, Memory, WriteBack)
Control 신호에 따른 각각의 역할 및 mux 사용
WriteBack에서 모든 pc값 변경
fib2.bin을 사용하여 jalr을 사용하였다.

4. Program Structure

main fuction에서 두 개의 while loop을 사용
전역 변수 선언
Open file

각각의 instruction을 memory에 저장

Fetch
Decode
Exe(control)
Memory
WriteBack

Fetch : Memory에 저장한 instruction을 하나씩 가지고 온다.
Decode : Fetch를 통해 가져온 instruction을 opcode를 보고 각각의 맞는 type에 따라 bit masking을 통해 구분해준다. (type : R, I, J)

Exe : Decode를 통해 각 type마다 가지고 있는 값을 사용하여 프로그램의 연산을 실행해준다.
control : Execute 할 때 처음 각각의 instruction opcode를 보고 control의 값들을 초기화한다.
Memory : Control에 따른 조건으로 sw와 lw 등에서 memory를 사용할 때 각각 결과를 출력해준다.
WriteBack : 모든 연산을 거친 후 결과 값은 다시 register에 저장하여야 한다.

5. Problems and solutions

처음 코드를 작성할 때 control을 사용하지 않고 작성하였다. 이 때 control을 사용하지 않음으로써 Memory를 사용하는 sw 나 lw 등에서 따로 Memory에 접근하지 않았다. 하지만 MemtoReg에 따라 ALU 연산 결과 값이 writeback하는지 Memory를 거쳐서 writeback을 하는지가 달라진다. 이처럼 Data path에서 control이 다양한 부분에서 조건으로 역할을 한다. 따라서 Data path에 맞게 작동할 수 있는 single cycle을 만들기 위해 control 부분을 조건문으로 만들어 주었다.
pc값 연산에서 코드를 디버깅하는 과정에서 값이 엉뚱한 곳으로 튀는 것을 발견하였다. 이럴 때 프로그램은 무한히 돌거나 아니면 이상한 곳에서 끝을 낸다. 이러한 이유를 찾아보니 몇몇의 예외들에서 오류가 발생하였다. 예외문들 이외에는 pc는 한번의 instruction이 지남에 따라 4씩 증가한다. 하지만 예외의 것들에서 pc값이 연산을 거친 후 다시 4가 증가할 때가 있었다. 이러한 문제들을 각각 예외문 처리를 해줌으로써 해결할 수 있었다.
처음 Data path를 볼 때 진행과정이 어떻게 가는지에 대해 확실한 이해가 없어 많은 어려움을 겪었다. 하지만 그림을 순차적으로 보면서 각각의 단계에 대해 나누어 생각하자 수월하게 이해하는데 많은 도움이 되었다. 또한 j, jal, Beq, Bne와 같은 pc값은 Data path의 그림을 위와 아래로 나누어 위쪽만 보면서 계산하자 좀더 이해하기 쉬웠다.
JALR 사용할 때 register 값이 업로드 되는 현상을 발견하였다. R type을 사용함에 있어서 writeback을 사용할 때 R type은 모두 값이 register값에 업로드 되었다. 이 때 jalr을 사용하면서도 이러한 업로드가 진행되어 예외문으로 처리해 주었다. 또한 fib.bin 파일도 instruction을 고쳐줘야했다. 이때 %!xxd를 사용하여 16진수 형으로 바꾸어 값을 입력하고 다시 나올 때 %!xxd -r을 사용하여 2진수인 bin파일로 변환해주고 나와야한다.

6. Build enviroment

Compilation : Linux Assam, with GCC
To compile : cd test_prog
gcc singlecycle.c
To run :
./a.out
input file : simple.bin, simple2.bin, simple3.bin, simple4.bin, fib.bin, gcd.bin, fib2.bin

7. Screen capture

```
65
66          // OPEN the file path
67          fp = fopen(path, "rb");
68          if(fp == NULL)
69          {
70                  printf("invalid input file : %s\n", path);
71                  return;
72          }
73          // in a loop,
74          // read file in from path
75          while(1)
76          {
77                  res = fread(&val, sizeof(val), 1, fp);
78                  inst = htonl(val);
79                  M[i] = inst;
80                  if(res == 0)
81                  {
82                          break;
83                  }
84                  i++;
85          }
86
87          while((pc/4) <= i && pc != 0xFFFFFFFF)
88          {
89                  cycle++;
90                  printf("----------------------------  %d  ----------------------------
    ---\n", cycle);
91                  printf(" pc : 0x%X\n\n", pc);
92                  Fetch();
93                  Decode();
94                  Exe();
95                  Memory();
96                  WriteBack();
97                  printf("next pc = 0x%X\n\n", pc);
98          }
99          printf("--------------------------------------------------------------------------------\n");
100         printf("Fianl : R[2] = %d, Cycle = %d\n\n", Reg[2], cycle);
101         // close the file
```

main문에서의 두 번의 while문 실행

```
368
369 void
370 Fetch()
371 {
372             inst = M[(pc/4)];
373 }
374
```

Fetch에서 pc는 4씩 증가함을 이용해 Memory에 저장된 instruction을 한 줄씩 가지고 온다.

```
375 void
376 Decode()
377 {
378         opcode = (inst & 0xFC000000) >> 26;
379         // R type
380         if(opcode == 0x0)
381         {
382                 rs = (inst & 0x3E00000) >> 21;
383                 rt = (inst & 0x1F0000) >> 16;
384                 rd = (inst & 0xF800) >> 11;
385                 shamt = (inst & 0x7C0) >> 6;
386                 funct = (inst & 0X3F);
387                 printf("0x%08X (opcode : 0x%X, rs : %d, rt : %d, rd : %d, shamt : 0x%X, f
    unct : 0x%X)\n", inst, opcode, rs, rt, rd, shamt, funct);
388         }
389         // J type
390         else if(opcode == 0x2 || opcode == 0x3)
391         {
392                 address = (inst & 0x3FFFFFF);
393                 JumpAddr = pc & 0xF0000000 | address << 2;
394                 printf("0x%08X (opcode : 0x%X, address : 0x%X, JumpAddr : 0x%X)\n", inst,
     opcode, address, JumpAddr);
395         }
396         // I type
397         else
398         {
399                 rs = (inst & 0x3E00000) >> 21;
400                 rt = (inst & 0x1F0000) >> 16;
401                 immediate = inst & 0xFFFF;
402                 BranchAddr = SignExt(immediate) << 2;
403                 printf("0x%08X (opcode : 0x%X, rs : %d, rt : %d, immediate : %d, BranchAd
    dr : 0x%X)\n", inst, opcode, rs, rt, immediate, BranchAddr);
404         }
405 }
```

Decode에서 각각의 type에 맞게 값을 초기화

```
317 void
318 Memory()
319 {
320         if(MemRead == 1)
321         {
322                 res = M[res];
323         }
324         if(MemWrite == 1)
325         {
326                 M[Index] = res;
327                 printf("M[Index] = %d\n", M[Index]);
328         }
329
330 }
331
```

Memory는 MemRead가 1이거나 MemWrite이 1일 때만 메모리에 접근하여 사용한다. 따라서 다음과 같이 조건문을 걸어 어떤 값을 writeback 시켜줄지 정한다.

```
348 void
349 WriteBack()
350 {
351         if(RegDst == 1)
352         {
353                 if(rd != 0)
354                 {
355                         if((opcode == 0x0) && (funct == 0x9))
356                         {
357                         }
358                         else
359                         {
360                                 Reg[rd] = res;
361                         }
362                 }
363         }
364         else
365         {
366                 if(rt != 0)
367                 {
368                         Reg[rt] = res;
369                 }
370         }
371
372         if((opcode == 0x2) || (opcode == 0x3))
373         {
374                 pc = JumpAddr;
375         }
376         else if(PCSrc2 == 1)
377         {
378                 pc = pc + 4 + BranchAddr;
379                 TB++;
380         }
381         else if(((opcode == 0x0) && (funct == 0x8)) || ((opcode == 0x0) && (funct == 0x9)))
382         {
383                 pc = res;
384         }
385         else
386         {
387                 pc = pc + 4;
388         }
389 }
390
```

WriteBack에서 RegDst를 통해 결과 값이 R[rd]와 R[rt]중 어디에 저장할지 정해준다.(Mux) 또한 j, jal, bne, beq, jr을 제외한 나머지는 pc 값이 4씩 증가하고 나머지는 각각의 조건에 맞게 pc 값이 변형된다. 이 때 Branch는 PCSrc2를 통해 값을 바꿀 수 있다.(Mux)

```
248 void
249 control()
250 {
251         if(opcode == 0x0) //R type
252         {
253                 RegDst = 1;
254         }
255         else
256         {
257                 RegDst = 0;
258         }
259         if((opcode != 0x0) && (opcode != 0x4) && (opcode != 0x5))
260         {
261                 ALUSrc = 1;
262         }
263         else
264         {
265                 ALUSrc = 0;
266         }
267         if(opcode == 0x23) //lw
268         {
269                 MemtoReg = 1;
270         }
271         else
272         {
273                 MemtoReg = 0;
274         }
275         if((opcode != 0x2B) && (opcode != 0x4) && (opcode != 0x5) && (opcode != 0x2) && (
    opcode != 0x3))
276         {
277                 RegWrite = 1;
278         }
279         else
280         {
281                 RegWrite = 0;
282         }
283         if(opcode == 0x23) //lw
284         {
285                 MemRead = 1;
286         }
287         else
288         {
289                 MemRead = 0;
290         }
291         if(opcode == 0x2B) //sw
292         {
293                 MemWrite = 1;
294         }
```

Control은 갂각의 Instruction에 대한 opcode를 확인하여 control signal의 값을 설정해주는 단계이다. 이를 통해 각각 Data path에 맞게 진행할 수 있다.

8. Personal feelings

이번 과제를 하면서 저번 계산기 과제를 할 때처럼 실수를 하지 않기 위해 미리 코드를 어떻게 짜야할지 큰 그림을 그리고 나서 진행을 하였다. 각각 어떠한 단계로 나눌지를 생각하고 순서에 맞게 코딩을 해나가자 Fetch, Decode, Execute 부분까지는 큰 문제없이 진행할 수 있었다. Bit masking은 교수님이 수업시간에 보여 주신 예제처럼 각각의 bit에 맞게 계산하여 값을 지정하였다. 또한 Mips green shit에서 이해가 가지 않는 문구는 교수님께 찾아가 질문을 드림으로써 해결할 수 있었다. 이후 다음 단계인 Memory와 WriteBack에서 많은 시간이 들었다. 처음 Control에 대한 이해와 Mux사용에 대한 그림이 잡히지 않아 며칠간 코딩에 대해 진도가 나가지를 않았다. 며칠간 생각하면서 생각을 정리한 후 각각의 경우에 맞게 값을 설정하는 함수가 있으면 좋겠다는 생각에 Control이라는 함수를 만들었다. Data path를 보면서 각각 type에 맞게 가지고 있는 값들을 설정해 주었다. 이를 Execute 함수에서 제일 먼저 실행해 줌으로써 ALU 연산이 진행되기 전 각각의 type에 맞게 control 신호를 설정할 수 있게 하였다. 이를 통해 Memory 부분에서도 각각 조건으로 mux를 사용할 수 있었다. 이번 single cycle과제를 하면서 Data path에 대한 이해도가 높아 졌고 다음에 진행하는 파이프라인에 대해서도 틈틈이 생각해보는 시간도 가질 수 있었다.

9. 결과 창

simple.bin

```
--------------------------------------------------------------------
Fianl : R[2] = 0, Cycle = 8
R-type : 4, I-type : 4, J-type : 0
Memory Access : 2
Taken Branches : 0
daehoon14@assam:~/test_prog$
```

simple2.bin

```
--------------------------------------------------------------------
Fianl : R[2] = 100, Cycle = 10
R-type : 3, I-type : 7, J-type : 0
Memory Access : 4
Taken Branches : 0
daehoon14@assam:~/test_prog$
```

simple3.bin

```
--------------------------------------------------------------------
Fianl : R[2] = 5050, Cycle = 1330
R-type : 409, I-type : 920, J-type : 1
Memory Access : 613
Taken Branches : 101
daehoon14@assam:~/test_prog$
```

simple4.bin

```
--------------------------------------------------------------------
Fianl : R[2] = 55, Cycle = 243
R-type : 79, I-type : 153, J-type : 11
Memory Access : 100
Taken Branches : 9
daehoon14@assam:~/test_prog$
```

fib.bin

```
--------------------------------------------------------------------
Fianl : R[2] = 55, Cycle = 2679
R-type : 818, I-type : 1697, J-type : 164
Memory Access : 1095
Taken Branches : 54
daehoon14@assam:~/test_prog$
```

gcd.bin

```
--------------------------------------------------------------------
Fianl : R[2] = 1, Cycle = 1061
R-type : 359, I-type : 637, J-type : 65
Memory Access : 486
Taken Branches : 45
daehoon14@assam:~/test_prog$
```

fib2.bin (jalr 사용)

```
--------------------------------------------------------------------
Fianl : R[2] = 55, Cycle = 2788
R-type : 927, I-type : 1806, J-type : 55
Memory Access : 1095
Taken Branches : 54
daehoon14@assam:~/test_prog$
```