|  |  |
| --- | --- |
| 2Pass Assembler 보고서 | |
| 학번 | 2019124206 |
| 이름 | 황현택 |
| 과목 | 시스템 프로그래밍 |
| 담당 교수님 | 최차봉 교수님 |

화살이(가) 표시된 사진

자동 생성된 설명

목차

1. 가이드라인
2. 2pass assembler 구현 방법
3. 코드 리뷰
4. 가이드라인

제가 구현한 어셈블러는 다음과 같습니다.

* Basic Level (10 점 : 전부 구현)

1. Data processing

Immediate value : rotate bit 4개와 immediate bit 8개로 표현 가능한 0 이상 수

Shift Immediate value : lsl, lsr, asr, ror 4가지 shift 명령어 + immediate value (ex : lsl #2)

Shift Register : lsl, lsr, asr, ror + registers ( ex : lsl r0 )

Opcode 종류 : mov, mvn, add, sub, rsb, eor (6개)

Cond 종류 : 기존에 있는 16개 모두

1. Swi

Swi + swi number

1. Mul

Mul rd, rm, rs, / mul rd, rm : 2개 또는 3개의 레지스터를 사용하도록 구현

Mla rd, rm , rs, rn : 무조건 4개의 레지스터를 사용하도록 구현

* Advanced level(4 점 : 일부 구현)

1. Symbol Table

Label과 명령어의 주소 값을 저장하는 table구현

이후 literal pool을 위해 .data 부분 symbol table도 구현

1. B 구현

B / bl + label 로 분기 구현

\* ldr/str/ldrb/strb는 구현하지 않았습니다.

* Challenging level(3 점 : 일부 구현)

1. pseudo instruction (ADR / LDR =)

adr r0, msg (.text section) -> add 또는 sub로 변하도록 (pc relative)

ldr r0, =0x10 (그냥 표현 가능한 값) -> mov로 변하도록

ldr r0, =256 ( rotate bit 사용으로 표현 가능한 값) -> mov로 변하도록

ldr r0, =0x12345678 ( 표현 불가능한 값 -> literal pool에 구현) -> ldr, pc relative로 변경

ldr r0, =msg1 ( .text에 있는 경우) -> adr과 비슷하게 offset을 이용한 pc relative로 변경

ldr r0, =msg2 ( .data에 있는 경우) -> 값을 literal pool에 구현

\*테스트 코드 참고 바랍니다. 진수 표현의 경우 일부 명령어에서만(ldr =) 0x 구현을 했습니다.

1. 2pass assembler 구현 방법 (작동 순서)

<1pass>

모든 명령어를 최종 결과 results에 저장합니다. 일반적인 명령어의 경우 1pass만으로 명령어를 완성시킬 수 있습니다. 예외로 label이 붙어있거나, ldr = (immediate) 명령어에서 immediate value가 rotate bit로 표현 불가능한 경우는 명령어의 일부를 results에 저장하고 이후 필요한 offset은 0으로 비워둡니다. 나머지 빈 부분은 2pass때 채우게 됩니다. 1pass를 돌면서 앞서 말한 2pass를 돌아야 하는 명령어의 경우, two\_pass\_line이라는 곳에 저장해 둡니다.

<Symbol table>

.text 섹션에서 label을 만날 경우, symbol table에 label과 현재 명령어의 주소를 함께 저장합니다.

<Data symbol table>

.data 섹션에 들어갈 경우, 이후 나오는 label은 데이터 형식이므로 data symbol table에 별도로 저장합니다. 이 때 데이터가 어디에 저장될 지 주소를 모르기 때문에 주소를 저장하는 것이 아닌, 그 데이터의 크기 만큼을 저장합니다. 예를 들어 .word 1234는 4, .asciz “hello”같은 경우는 6(5byte + 마지막 zero)를 저장합니다.

<Literal pool>

Label 뒤에 데이터가 나온 경우, 표현 불가능한 값이라면 literal pool에 label을 저장합니다. 이 때, 그에 해당하는 데이터가 나올 주소는 모르기 때문에 0으로 채워둡니다.

<2pass>

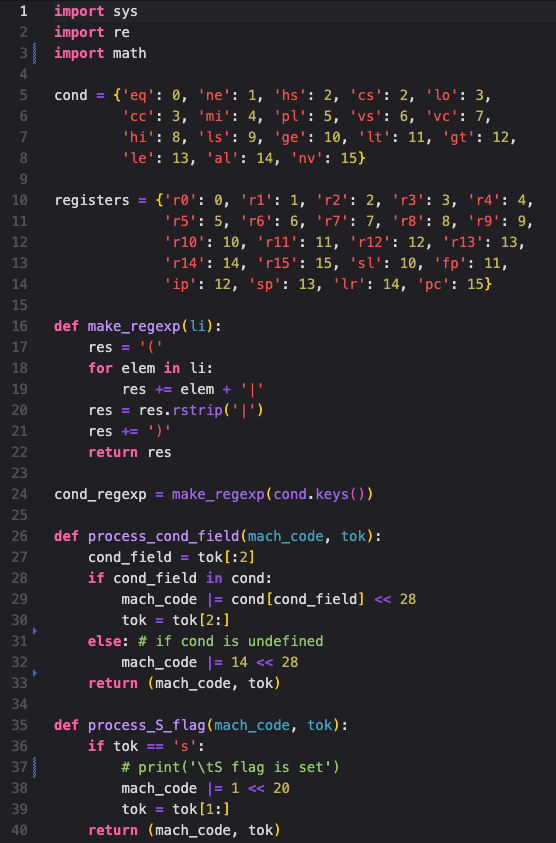
1pass에서 저장해 둔 two\_pass\_line을 순회합니다. 일반적인 명령어는 2pass를 돌 필요가 없기 때문에 일부만 다시 돌면 됩니다. 2pass를 돌아야 하는 경우는 label을 사용하거나 literal pool을 알아야 할 때 입니다. 단순한 b, bl 명령어이거나 adr 명령어의 경우, symbol table을 활용해 현재 pc와 target간의 offset을 구해서 명령어를 채워줍니다.

Ldr =label 명령어의 경우, 처음에 symbol테이블을 살핀 다음 데이터가 있다면 adr과 똑같이 처리합니다. Symbol table에 없을 경우 data\_symbol\_table을 살펴 봅니다. Data\_symbol\_table에 있는 주소를 바탕으로 results에 새로운 명령어를 추가합니다. 이 작업은 어셈블러가 .text 뒤에 .ltorg를 붙이는 과정입니다.

Ldr =immediate일 경우, literal pool에 그 값을 저장하고, 똑같이 results 뒤에 붙여줍니다.

모든 작업이 끝나면 results를 최종 출력해줍니다.

1. 코드 리뷰



Condition과 registers의 선언, condition flag와 s flag를 넣어주는 부분입니다.

이로써 명령어에 condition flag, s flag를 세팅해줍니다.

예제와 달라진 부분은 없습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Main에서 이 함수를 호출하면 명령어 별 mach\_code를 설정해서 리턴합니다.

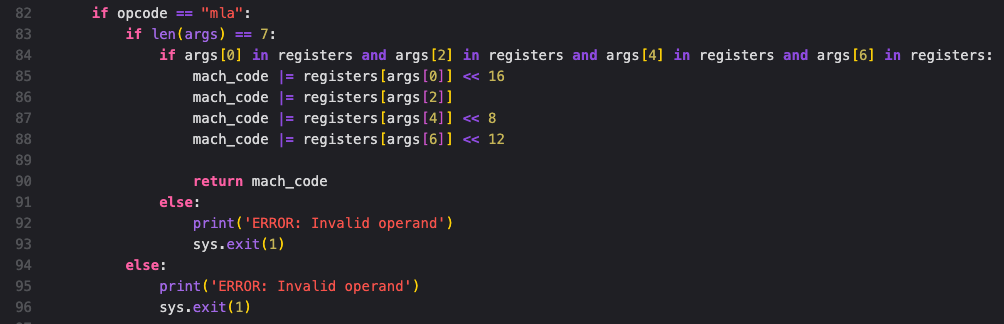
특이한 명령어인 swi를 먼저 처리했습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Mul 명령어 부분입니다. 레지스터를 받을 수 있는 개수가 2개 또는 3개이기 때문에,

Args의 길이로 판단했습니다.



Mla 명령어 부분입니다. Mla의 경우 무조건 4개의 레지스터가 와야 하기 때문에 먼저 처리했습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

일반적인 명령어에 들어가기 앞서, ISA문법에 맞지 않는 경우를 예외처리 해주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 일반적인 명령어 6개에 대한 처리입니다. 명령어는 rn, rs의 유무, lsl 과 같은 shift 연산자의 유무에 달라지기 때문에, args길이에 따라 처리해주었습니다.

길이가 3인 경우는 add, sub, rsb, eor 이 가능합니다. Add r0, r1과 같은 경우입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

길이가 5인 명령어는 add, sub, rsb, eor이 가능합니다. Add r0, r1, r2과 같은 경우입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

길이가 6인 경우, add, sub, rsb ,eor 이 가능합니다.

Add r0, r1, lsl #3과 같은 경우입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 길이가 8인 경우는 add, sub, rsb, eor이 가능합니다.

Add r0, r1, r2, lsl #2과 같은 경우입니다.

모든 명령어 처리가 끝났기 때문에 mach\_code를 리턴합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Main 함수에서 line을 읽고 실행하는 함수입니다.

명령어 종류별로 선언을 해두고 아래에서 명령어 별 기본 처리를 해주었습니다. 텍스트이(가) 표시된 사진

자동 생성된 설명

Mov와 mvn입니다. Opcode를 넣어주고 condition flag, s flag를 세팅하고process2 작업을 해 줍니다.

이러한 방식과 똑같이 add / sub / rsb / eor도 작업하므로 넘어가도록 하겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

B / bl 명령어는 특이합니다. 1pass 만으로는 처리할 수 없기 때문에 condition flag만 세팅하고 two\_pass\_line에 넣어줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Ldr= format도 1pass만으로는 할 수 없습니다. 또한 어떤 값이 오는가에 따라 mov 로 대체해야 하기 때문에 rotate bit 이 가능한지 try catch문으로 검사합니다. 여기서 나오는 rotateIm 함수가 그 역할을 해 줍니다. 이후에 설명하겠습니다. 값이 표현 불가능하다면 literal pool에 넣어야 하기 때무에 data symbol table에 넣어줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Adr 또한 1pass 만으로 알 수 없기 때문에 two pass line에 넣어줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Immediate value를 rotate bit를 사용한 12bit 표현으로 바꿔주는 함수입니다. 일반적인 경우 32bit 내에서 연속된 8bit로 표현 가능하다면 12bit로 표현할 수 있기 때문에 result를 구하는 건 간단합니다. 이제 다음 장에서 일반적이지 않은 경우를 보겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

일반적으로 연결된 8bit의 시작은 왼쪽, 끝은 오른쪽입니다. 하지만 시작이 오른쪽이고 끝이 왼쪽이면서 32bit의 끝과 끝이 연결된 경우 또한 총 8bit이내로 사용한다면 표현이 가능합니다. 따라서 32bit의 가운데인 16번째 bit에서 각각 좌, 우로 가면서 1이 처음으로 나오는 지점들을 구합니다. 이후 구한 지점에서 길이를 제서 8bit 이하로 사용한다면 shift를 이용해 12bit로 결과값을 만들어 반환합니다.

조심해야 할 부분은 정확히 8bit만 사용한다고 해서 다 되는 것은 아닙니다.

1110 0000 0000 0000 0000 0000 0001 1111같은 경우 총 8bit만 사용하지만 표현 불가능합니다. 이를 shift하게 되면

0000 0000 0000 0000 0000 0001 1111 1110으로 총 9bit를 사용한 꼴이 되기 때문입니다. 이러한 경우를 생각하면서 구현해주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

프로그램의 시작과 결과값을 초기화합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

모든 line을 돌면서 line의 시작에 따라 경우를 나눠줍니다. 처음으로 label이 올 경우입니다.

.text 섹션에서 label이 온 후, .word가 오는 경우와 .asciz가 오는 경우, 아무것도 안 나오는 경우로 나누었습니다. .text 섹션은 symbol 테이블에 label과 그 주소를 저장해줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

.asciz의 경우도 .word와 비슷하게 했지만, 문자열에 escape 문자가 오는 경우가 문제가 되었습니다. Escape 문자 수를 세 주고, 마지막에 zero까지 더해서 string을 완성시켰습니다. 또 하나의 문제로 총 길이가 4의 배수이면 주소가 4만큼 증가하지만, (4의 배수) +2의 경우도 align문제가 발생하지 않기 때문에 이 부분을 신경써주어서 구현했습니다.

나머지 1 또는 3같은 홀수의 경우 align 문제로 종료시켰습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

.text 섹션의 마지막입니다. Word나 asciz같은 데이터가 아니라면, call형태이기 때문에 symbol 테이블에 주소값만 저장해주었습니다.

또한 label로 시작하지 않는다면 일반적인 명령어 (mov, sub같은) 이기 때문에 process\_instruction 함수를 호출했습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후는 .data 섹션입니다. 여기서 label이 나올 경우는 data symbol table에 저장을 해주었고,

마찬가지로 word와 asciz등의 데이터 형식이 올 수 있기 때문에 위와 비슷하게 구현해주었습니다.

조금 다른 점은 .data에 저장되는 데이터는 나중에 저장될 주소를 정확하게 알 수 없기 때문에 주소를 저장하는 것이 아닌 데이터의 길이를 구해서 저장해주었습니다.

또한 literal pool에 label을 저장하는데, 정확한 주소는 모르기 때문에, 0을 채워주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

2pass에 들어가기 앞서, data symbol table에 저장된 데이터의 길이를 가지고 테이블을 업데이트 시켰습니다. Ltorg 영역에 저장될 때, label의 경우 그 데이터의 길이만큼 확보된 후 데이터가 저장되기 때문에, before에 현재 길이인 cost를 합친 만큼을 다음 데이터 길이에 넘겨주면서 지정해주었습니다.

또한 ltorg 영역에 들어가기 전 명령어의 주소 값을 4의 배수로 맞춰주었습니다. 텍스트, 화면이(가) 표시된 사진

자동 생성된 설명

2pass 부분입니다. 1pass에서 저장해 두었던 2pass line을 돌면서 명령어에 따라 빈 부분을 채워넣어 주었습니다.

B / bl 명령어의 경우 symbol 테이블에서의 target 주소와, 현재 pc값 사이의 offset을 채워넣어 주었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Ldr = format명령어 입니다. Ldr의 경우 뒤에 immediate가 온다면 그 값이 rotate bit로 표현 가능한 값인지 확인을 하고, 표현 가능하다면 mov로 명령어를 변경해줍니다. 그게 아니라면 literal pool을 활용해서 상대 주소값을 이용한 pc relative 방식으로 표현해줍니다. 이후 ltorg 부분을 구현하기 위해 결과 맨 뒤에 그 값을 저장해줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Ldr r0, = label이 올 경우입니다. 처음에 label을 symbol 테이블에서 찾아보고, 있다면 pc relative로 바로 변경해줍니다. 없다면, data symbol table를 확인하고 literal pool에 값을 저장합니다.

Target과 pc의 offset으로 비어있는 명령어를 채워주고, ltorg에 추가하기 위해 결과 맨 뒤에 데이터를 추가해줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 adr 명령어입니다. Adr 뒤에 오는 값이 immediate로 표현 가능하다면 add 또는 sub 명령어로 바꿔줍니다. 또한 ldr과 마찬가지로 pc와 target의 offset을 구해서 results를 채워줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

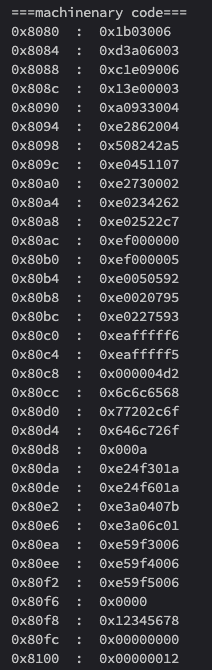
마지막 결과 출력 부분입니다. 우선 16진수 형태로 모든 명령어 + ltorg 부분을 출력해주고, 이해를 돕기 위해 symbol table과 data symbol table, literal pool을 추가로 출력해주었습니다.

테스트 코드

텍스트이(가) 표시된 사진

자동 생성된 설명

최종 결과값(제작한 어셈블러) qemu 결과값

텍스트이(가) 표시된 사진

자동 생성된 설명