Architektury systemów komputerowych

Projekt nr 1

Asembler dla architektury MIPS

Termin oddawania: 1 czerwca 2018

Przygotowanie

Przed przystąpieniem do implementacji projektu należy zapoznać się z treścią rozdziału „Assemblers, Linkers, and the SPIM Simulator" książki „Computer Organization and Design: The Hardware/Software Interface" oraz zestawem instrukcji opisanym w MIPS Reference Data Card[1]. Gruntowny przegląd zagadnień niskopoziomowych dotyczących architektury MIPS można znaleźć w dokumencie „MIPS Assembly Language Programmer's Guide" dostępnym na stronie przedmiotu.

Wymogi formalne

Dopuszcza się użycie następujących języków programowania: C, C++, Scheme lub Python. Pliki źródłowe oraz **testy** należy spakować do archiwum w formacie «tar.gz» o nazwie «indeks_imie_nazwisko.tar.gz» i umieścić je w systemie SKOS. Po rozpakowaniu projekt ma mieć następującą strukturę katalogów:

```
999999_jan_nowak/
  README.md
  Makefile
  mipsasm.c
  lexer.c
  line.h
  relo.h
  ...
  tests/
    test1a.in
    test1b.in
    test2a.in
    ...
```

Oceniający zadania używa komputera z zainstalowanym systemem Debian GNU/Linux 9 dla architektury x86-64. Ściąga z systemu SKOS archiwum dostarczone przez studenta, po czym sprawdza poprawność struktury katalogów. Następnie czyta plik README.md z uwagami od studenta i jeśli to potrzebne buduje projekt używając pliku Makefile. Po zapoznaniu się z kodem źródłowym uruchamia plik wykonywalny asemblera z testami dostarczonymi przez studenta.

UWAGA! Pokrycie kodu testami[2] jest brane pod uwagę przy ocenie rozwiązania.

Opis

Celem listy jest zbudowanie uproszczonego asemblera dla architektury MIPS, dzięki czemu dowiemy się jak działają ostatnie etapy kompilacji dla języków wyższego poziomu. Zakładamy, że porządek bajtów jest określony według konwencji *Little Endian*. Kodowanie instrukcji ma być zgodne z MIPS ISA. W razie wątpliwości można podglądać co robi środowisko MARS[3].

[1] http://booksite.elsevier.com/9780124077263/downloads/COD_5e_Greencard.pdf
[2] https://en.wikipedia.org/wiki/Code_coverage
[3] http://courses.missouristate.edu/kenvollmar/mars/

UWAGA! Należy gruntowanie przemyśleć strukturę programu! Inaczej będzie ciężko zapanować nad kodem!

Wskazówki: Zanim przystąpisz do implementacji **przeczytaj wszystkie podpunkty**! Pierwszy przebieg asemblera tworzy reprezentację całego pliku wejściowego w postaci listy rekordów opisujących linie. Drugi będzie tworzyć zawartość sekcji «.text» i «.data». Sekcja to po prostu kontener na ciąg bajtów. Oprócz tego będziemy przypisywać etykietom adresy i tworzyć listę relokacji – miejsc, które należy odwiedzić w trzecim przebiegu asemblera, aby obliczyć i wstawić referencje do etykiet.

Zadanie 1 (5). Zaimplementuj asembler tłumaczący instrukcje arytmetyczne procesora MIPS na kod maszynowy. Asembler przetwarza plik tekstowy składający się z linii w następującym formacie:

```
instrukcja    operand1,operand2,...    # komentarz
```

Komentarz jest opcjonalny, a ilość operandów jest zależna od instrukcji. Na razie operandami są wyłącznie rejestry: $0...$31 i ich odpowiedniki symboliczne np.: $v0, $ra, ... oraz stałe całkowitoliczbowe. Instrukcje, które musi rozpoznawać asembler to: «lui», «addi», «addiu», «slti», «sltiu», «andi», «ori», «xori», «lui», «sll», «srl», «sra», «sllv», «srlv», «srav», «mfhi», «mthi», «mflo», «mtlo», «mult», «multu», «div», «divu», «add», «addu», «sub», «subu», «and», «or», «xor», «nor», «slt», «sltu».

Na wyjście należy wydrukować listing programu, tj. zawartość pliku źródłowego, gdzie do każdej linii zawierającej instrukcję należy dołączyć w formie szesnastkowej jej adres i zapis binarny, np.:

```
.text
00000000    02328020    add     $s0,$s1,$s2
00000004    2268FFF4    addi    $t0,$s3,-12
```

Wskazówki: Postaraj się zaprojektować swój asembler tak, aby był łatwo rozszerzalny. Pisanie kodu do przetwarzania każdej instrukcji z osobna jest niemądrym pomysłem. Lepiej zaprojektować format metadanych opisujących kodowanie instrukcji, np.:

```
add:reg,reg,reg:R:0/%0/%1/%2/0/32
addi:reg,reg,sint16:I:8/%1/%0/%2
```

Zadanie 2 (2). Zaimplementuj możliwość stosowania dyrektyw, których znaczenie jest objaśnione w §A.10 podręcznika. Dyrektywy, które musi obsługiwać asembler to: «.align», «.ascii», «.asciiz», «.byte», «.data», «.half», «.space», «.text», «.word» i «.org». Dyrektywy wpisują dane pod miejsce wskazywane przez kursor po czym go zwiększają, albo przenoszą kursor na koniec innej sekcji. Zauważ, że dyrektyw «.text» i «.data» można używać wielokrotnie. Odpowiednio zmodyfikuj listing, tj.:

```
.data
00008000                                    .org      0x8000
00008000    68656c6c6f20776f726c640a00      .asciiz   "hello world!\n"
0000800e    0000                            .align    4
00008010    c0decafe                        .word     3235826430
```

Wskazówki: Powyższy format dla każdej linii wejściowej można utworzyć przechowując w rekordzie linii odnośnik do zakodowanego ciągu bajtów.

Zadanie 3 (3). Dodaj obsługę instrukcji skoków: «beq», «bne», «j», «jal», «jr». Należy umożliwić stosowanie etykiet, czyli symbolicznego zapisu adresów. Etykieta to ciąg znaków opisany wyrażeniem regularnym «[A-Za-z][A-Za-z0-9_]*». Możliwe są następujące typy relokacji:

- MIPS_PC16: adres relatywny skoku do etykiety dla instrukcji «beq» i «bne»,
- MIPS_LO16: operand «%lo(etykieta)» – młodsze 16 bitów adresu docelowego,
- MIPS_HI16: operand «%hi(etykieta)» – starsze 16 bitów adresu docelowego,
- MIPS_26: adres absolutny skoku do etykety dla instrukcji «j» i «jal»,
- MIPS_32: argument dyrektywy «.word».

Zbuduj tablicę symboli, gdzie każdy element jest krotką «`symbol, nr sekcji, offset`» i wyświetl ją w listingu programu. Należy również utworzyć i wyświetlić tablicę relokacji «`.rela.text`» i «`.rela.data`». Dla przypomnienia – relokacja to krotka «`adres, typ, nr sekcji, nr symbolu, offset`».

Zadanie 4 (2). Zaimplementuj w swoim asemblerze obsługę następujących pseudoinstrukcji: «`blt`», «`bgt`», «`li`», «`la`» i «`move`». Niektóre z nich mogą być kodowane do więcej niż jednej instrukcji podstawowej, inne wymagają rozbicia adresu lub stałej symbolu na dwie 16-bitowe liczby. Pseudoinstrukcja «`li`» musi akceptować 32-bitową stałą, a «`la`» absolutny adres etykiety.

Wskazówki: Podobnie jak w punkcie pierwszym, dobrze jest wymyśleć jakieś metadane służące do opisu pseudoinstrukcji np.: "`blt:reg,reg,label:P:slt $at,%0,%1;bne $at,$0,%2`". Przed pierwszym przebiegem będzie trzeba uruchomić procedurę rozwinięcia pseudoinstrukcji. Zauważ, że taka funkcjonalność jest zaczynem implementacji makroasemblera[4].

Zadanie 5 (2). Rozszerz swój asembler o możliwość stosowania instrukcji operujących na pamięci. Trudnością w tym punkcie jest dodanie operandów reprezentujących tryby adresowania: «`($a0)`» i «`offset($a0)`». Nowe instrukcje, które należy wspierać to: «`lb`», «`lbu`», «`lh`», «`lhu`», «`lw`», «`lwu`», «`sb`», «`sh`», «`sw`».

[4] `https://en.wikipedia.org/wiki/Assembly_language\#Macros`