Електротехнички факултет

Универзитет у Београду

Домаћи задатак јун/јул 2015

Системски софтвер

ИНТЕРПРЕТАТИВНИ ЕМУЛАТОР

Студент:

Живановић Ђорђе

2013/0033

Професор:

Стојановић Саша

Београд, Јун 2016

# Задатак (15 бодова)

Нaправити интерпретативни емулатор за рачунар описан у прилогу. Улаз емулатора је излаз асемблера, при чему је могуће задати већи број предметних програма које је потребно учитати, повезати и покренути. Називи предметних програма се задају као аргументи командне линије. Емулација је могућа само уколико након учитавања и повезивања нема неразрешених симбола. Приликом покретања емулације, као први аргумент командне линије задаје се и фајл којим се оређује распоред секција приликом пуњења програма у меморију (скрипт). Формат фајла за управљање пуњењем и поступак пуњења описани су у наставку. У сваком реду датотеке записан је назив једне секције или наредба за доделу вредности. Понављање назива секција није дозвољено. Уколико се наредбама за доделу вредности не уради другачије, секције треба сместити редоследом по којем су наведене у скрипту, једну за другом. Секције које нису наведене у скрипту сместити после последње секције наведене у скрупту и то по абецедном редоследу. У скрипту је могуће користити и посебан симбол (тачка, ".") који представља текућу адресу. На почетку рада, вредност овог симбола је 0. Када се у скрипту наиђе на секцију, секција се смешта почев од текуће адреса, а текућа адреса се увећава за величину поменуте секције. Текућу адресу је могуће променити и тиме променити распоред секција у меморији. Текућа адреса се мења тако што се симбол тачка појави са леве стране доделе вредности ("="). Нова вредност не сме бити мања од претходне (у супротном пријавити грешку и прекинути процес). Вредност је могуће доделити и другим симболима, тако што се назив симбола појави са леве стране доделе вредности. Тако дефинисан симбол је могуће користити равноправно са осталим глобалним симболима дефинисаним у предметним програмима који се покрећу (нису дозвољене вишеструке дефиниције). Вредност која се појављује са десне стране је прост израз у којем учествују константе и други до тада дефинисани симболи. Операције које су дозвољене су сабирање, одузимање и поравнање. Поравнање се записује попут позива функције која прихвата два аргумента. Први аргумент је полазна адреса, док је други број делилац који се користи за поравнање. Операција align враћа најмању адресу дељиву задатим делиоцем која није мања од адресе задате првим аргументом.

# Решење

Задатак је решен кроз различите структуре где су симулирани регистри процесора, меморија,и табела прекидних рутина. Прво се читају фајлови и поново праве табеле симбола, релокационе табеле, секције, и све што треба. Затим се парсира loadScript и и секције пуне у меморију, тј. ажурирају почетне адресе и глобални симболи дефинисани у ld скрипти додају као глобални. Затим се разрешавају сви .extern симболи. Ако постоји неразрешени симбол баца се грешка. Онда се секције сортирају по почетним адресама и после по именима. На крају се разрешавају релокације и потом пуни програм у меморију. Након тога иницијализује се улазна тачка програма (ако нема main пријави се грешка), регистри, стање psw-a… Након тога се покрене програм, с тим да је тајмер активан као и тастатура. Након парсирања сваке асемблерске инструкције ради се обрада прекида, где се прекид извршава у зависности од предности. Предност имају они са мањим бројем. Програм се завршава кад се наиђе на инструкцију int 0x0.

**Коришћене структуре**

* **CPU** је класа која врши декодовање инструкција, прекида и емулирање истих
* **Emulator** је класа која се бави линковањем свих фајлова и пуњењем извршног у меморију, ако се деси нека грешка при линковању програм се прекида, на крају ако нема грешке треба да испише sucessfull
* **EmulatorError –** садржи типове грешака и одговарајуће исписе и кодове
* **IVT –** садржи иницијализацију IVT табеле и одговарајућих прекидних рутина
* **Memory** садржи структуре неопходне за симулацију меморије и улазно излазне меморије
* **RellocationTableEntry** представља релокационих записа за одговарајућу релокацију
* **RellocationTable** представља табелу релокационих записа за одговарајућу секцију
* **SymbolTableEntry** садржи одговарајући глобални симбол који ће бити видљив у фази линковања
* **SymbolTable** садржи све глобалне симболе који ће бити видљиви у фази линковања
* **Section** садржи податке о секцијама, величину, почетну адресу...
* **Tokenizer** је класа која се бави парсирањем текста и пријављивањем грешака ако дати формати нису испуњени
* **Main** улазна тачка програма, налази се учитавање објеткинх фајлова, линковање, пуњење у меморију и извршавање, ако је све протекло у реду

**Тестирање**

Програм је преведен коришћењем команде:

g++ -std=c++0x -o Emulator CPU.cpp Emulator.cpp IVT.cpp Main.cpp Memory.cpp RellocationTable.cpp RellocationTableEntry.cpp Section.cpp SymbolTable.cpp SymbolTableEntry.cpp Tokenizer.cpp –pthread

И одговарајући извршни фајл је пребачен у одговарајући тест пример и тестиран излаз.

**Тестови**

* **Тест1**

Приказује како се користећи рекурзија исписује број. Приказано је и да је распоред littleEndian у меморији. Да се исправно ажурирају LR, PC, и и да су типови адресирања покривени. 123456 у овом случају испише

* **Тест2**

Приказује симултани ради тајмера и тастатуре. Ако се брзо буду куцала дугмад на тастатури може се уочити да ће тајмер имати предност. Успут је дат и пример коришћења директиве .align. Исписаће оно шта корисник буде откуцао кад притисне enter. Кад наиђе на слово k после тога прекида програм.

* **Тест3**

Приказује употребу већине инструкција у асемблеру и како се исправно линкује програм иако је у другом фајлу дефинисана функција за испис броја у бинарном формату.

* **Тест4**

Приказује како се исправно формира објектни фајл, кад су дати сви типови релокационих записа и сви типови директива и на крају и извршни фајл.

* **Тест5**

Приказује како се исправно парсирају сви типови инструкција. Али грешка настаје у linkage делу због недефинисаног симбола a

* **Тест6**

Приказује како се исправно ажурирају битови и ако се направи испремештан садржај у ld скрипти приказује да емулатор добро распореди и изврши програм. Исписаће се битови одговарајућег броја (0xfedcab98) у бинарном запису и то два пута. Затим ће се исписати адреса .text.2 секције у бинарном (…0101000) што показује коришћење align у скрипити. 1010100 затим што показује адресу следеће секције (у нашем случају data.print). Затим исписује disp који је дефинисан у ld.txt скрипти, што показује да је могуће користити глобалне променљиве унутар кодова. Затим адресу main, што је 1010100 + 24 = 1101100