การใช้งาน ARC Simulator ในการศึกษา Instruction Set

อาจารย์ปิยะพงษ์ แดงขำ

สาขาวิชาวิศวกรรมสารสนเทศและการสื่อสาร

มหาวิทยาลัยราชภัฏเทพสตรี

บทคัดย่อ

ARC Simulator ถูกสร้างขึ้นเพื่อประกอบการเรียนการสอน Computer Architecture and Organization โดยอ้างอิงจากหนังสือ Computer Architecture and Organization: An Integrated Approach ของ Miles J. Murdocca และ Vincent P. Heuring ซึ่งสามารถแสดงให้เห็นถึงการเขียน Instruction Set ควบคุมการทำงานของคอมพิวเตอร์ โปรแกรมถูกออกแบบมาให้ใช้งานได้ง่าย และสังเกตการณ์เปลี่ยนแปลงของแต่ละรีจิสเตอร์ได้ด้วย

คำสืบค้น

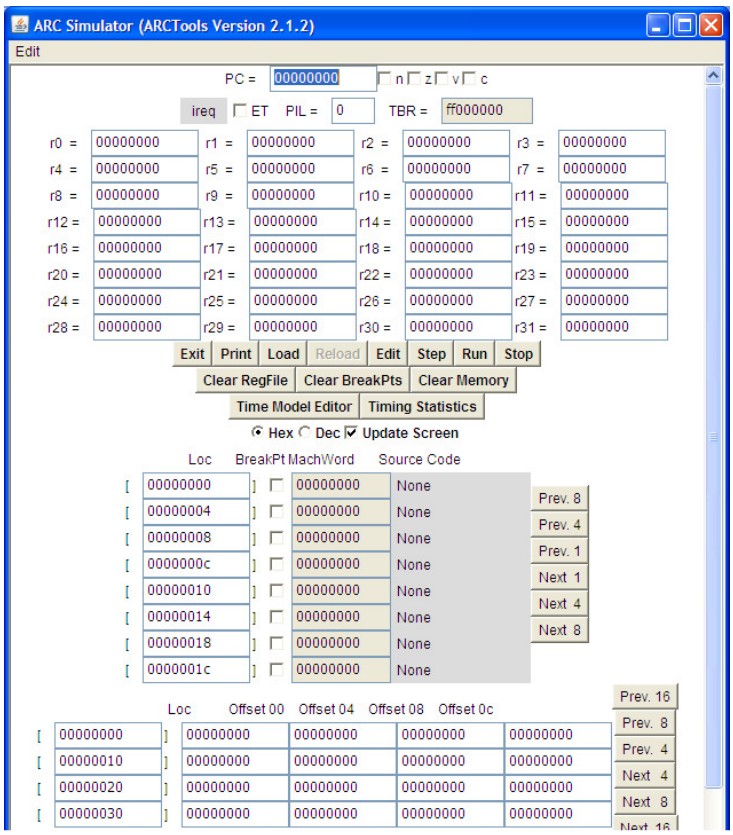
ARC, Instruction Set

การเรียกใช้ ARC Simulator

ไฟล์โปรแกรมจะอยู่ในรูปแบบของ jar ซึ่งเขียนด้วยภาษาจาวา ผู้ใช้จะต้องมี Java Runtime Environment จึงจะสามารถทำงานได้ โดยเรียกใช้ดังนี้

java -jar ARCToolsv2.1.2.jar

หลังจากที่เรียกใช้งานโปรแกรมแล้วจะพบหน้าต่างการทำงานดังนี้

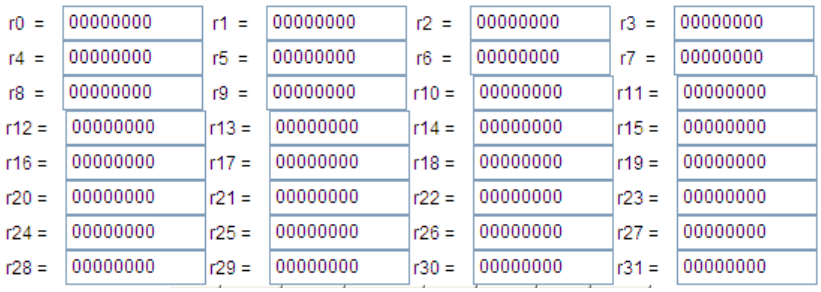


รูปที่ 1 หน้าต่างโปรแกรม ARC Simulator

ส่วนประกอบของโปรแกรม

รายละเอียดรีจิสเตอร์ต่างๆ

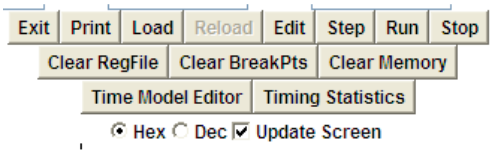
โปรแกรมจำลองจะแสดงรายละเอียดของรีจิสเตอร์ตั้งแต่ r0 ถึง r31 เพื่อให้รู้ค่าที่ถูกเก็บไว้ในหน่วยความจำ โดยจะสัมพันธ์กับ Instruction ที่ใช้ นักศึกษาสามารถสังเกตการเปลี่ยนแปลงการทำงานได้โดยอาศัยการแสดงค่ารีจิสเตอร์จากส่วนนี้



รูปที่ 2 ส่วนรายละเอียดรีจิสเตอร์

กลุ่มคำสั่งควบคุมการทำงาน

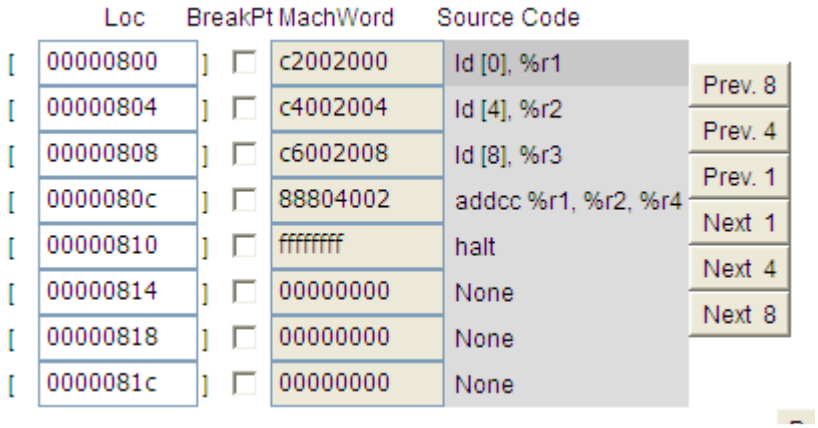
เป็นส่วนที่ผู้ใช้งานจะใช้งานในการสร้างชุดคำสั่ง รวมถึงให้ชุดคำสั่งทำงาน ซึ่งจะสามารถสั่งให้ทำงานจนครบชุดคำสั่ง หรือให้ทำงานเป็นลำดับขั้นตอนทีละคำสั่งก็สามารถทำได้ นอกจากนี้ยังเป็นส่วนที่กำหนดค่าการแสดงผลของรีจิสเตอร์ว่าให้แสดงเป็นรูปแบบฐานสิบหรือฐานสิบหก



รูปที่ 3 ปุ่มควบคุมการทำงานโปรแกรม

รายละเอียดชุดคำสั่ง

นักศึกษาสามารถสังเกตการทำงานแต่ละขั้นตอนโดยการกดที่ปุ่ม Step ซึ่งจะเห็นการทำงานเป็นลำดับตามชุดคำสั่งแต่ละบรรทัด การทำงานจะเป็นไปตาม instruction ที่กำหนด และสามารถดูการเปลี่ยนแปลงค่าได้ที่หน้าต่างรีจิสเตอร์



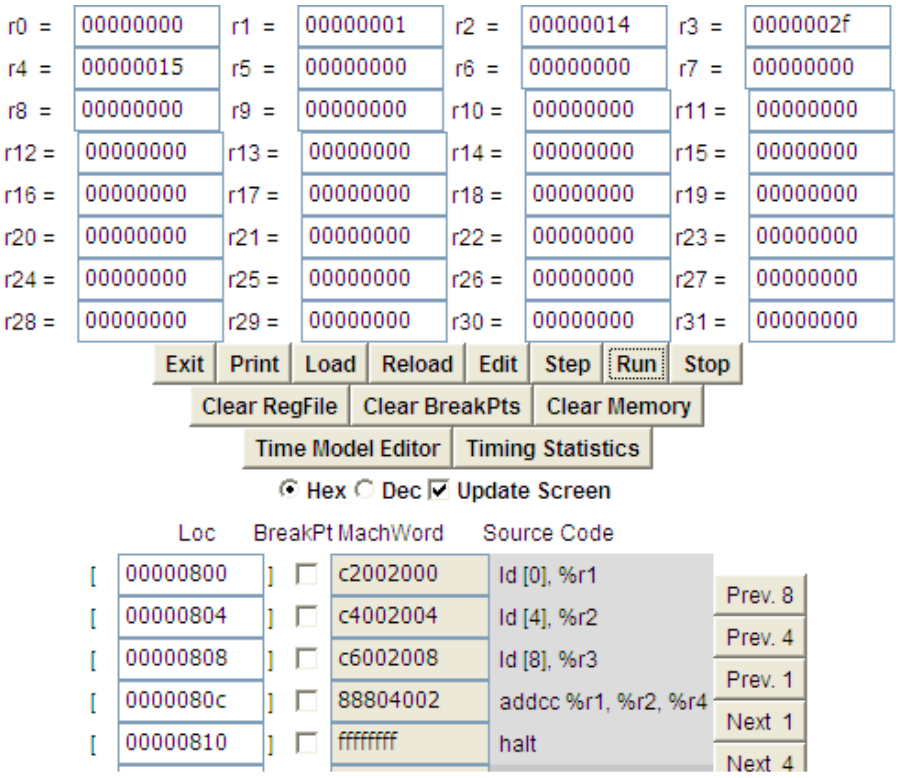
รูปที่ 4 ส่วนแสดงชุดคำสั่ง

การเขียนโปรแกรมชุดคำสั่ง

ให้ทำการกดที่ปุ่ม Edit ในส่วนของกลุ่มคำสั่งควบคุมการทำงาน จะปรากฏหน้าต่างใหม่ว่างๆ ขึ้นมา แล้วให้นักศึกษาสร้างชุดคำสั่งดังนี้

```
.begin
      one:          1
      two:          20
      three:        0x2F
      .org 2048
a_start      .equ 3000
      ld [one], %r1
      ld [two], %r2
      ld [three], %r3
      addcc %r1,%r2,%r4
      halt
.end
```

หลังจากนั้นให้นักศึกษาทำการแปลชุดคำสั่งด้วยการกดปุ่ม Assemble โปรแกรมจะแสดงผลการแปลชุดคำสั่ง พร้อมทั้งแสดง error ถ้ามีการเขียนคำสั่งผิดพลาด และให้กดปุ่ม Bin->Sim เพื่อทำการโหลดข้อมูลเข้าสู่โปรแกรมจำลอง โดยให้นักศึกษากด Run ที่หน้าต่างโปรแกรมจำลอง แล้วสังเกตการเปลี่ยนแปลงของค่ารีจิสเตอร์



รูปที่ 5 ผลการจำลองการทำงานของชุดคำสั่ง

การอ้างอิงตำแหน่งหน่วยความจำ

การกำหนดค่าตัวแปรที่ใช้ใน ARC Simulator อาจจะทำได้โดยการใส่ label หรืออ้างอิงผ่านตำแหน่งที่เก็บข้อมูลไว้ โดยการคำนวณค่าได้จากตำแหน่งเริ่มต้นที่กำหนดในการเขียนโปรแกรม ไปจนถึงตำแหน่งที่จัดเก็บตัวแปรนั้นๆ

ตำแหน่งของ Instruction

ให้ทำการศึกษาโปรแกรมดังนี้

```
      .begin
      .org 2048
main: ld    [x], %r1
      ld    [y], %r2
      addcc %r1, %r2, %r3
      st    %r3, [z]
      jmpl  %r15 + 4, %r0
x:    15
y:    9
z:    0
      .end
```

เมื่อทำการ Assemble แล้ว ให้ทำการกด Show Lst File จะปรากฏรายละเอียดการแปลภาษา โดยจะแสดง Instruction แต่ละบรรทัด พร้อม address ที่เก็บไว้ในหน่วยความจำ ซึ่ง Instruction แรกจะเริ่มที่ตำแหน่ง 2048 ตามที่ได้กำหนดไว้ด้วยคำสั่ง .org หลังจากนั้นจะบวกเพิ่มไปทีละ 4 ไบต์ (ขนาดของ 1 word) และในส่วนท้ายจะมี Symbol Table สรุปค่าสัญลักษณ์ตาม address ต่างๆ

```
(ARCTools Version 2.1.2)
 HexLoc     DecLoc    MachWord   Label        Instruction                Comment
                                              .org 2048
00000800  0000002048  c2002814  main:        ld [2068], %r1
00000804  0000002052  c4002818               ld [2072], %r2
00000808  0000002056  86804002               addcc %r1, %r2, %r3
0000080c  0000002060  c620281c               st %r3, [2076]
00000810  0000002064  81c3e004               jmpl %r15, 4, %r0
00000814  0000002068  0000000f  x:
00000818  0000002072  00000009  y:
0000081c  0000002076  00000000  z:

--- Symbol Table ---
x: 2068
main: 2048
z: 2076
y: 2072
```

การอ้างอิงตัวแปร

จากโปรแกรมข้างต้น เมื่อต้องการอ้างอิงค่าตัวแปรจะทำด้วยคำสั่ง ld ดังนี้

```
ld      [x], %r1
```

ซึ่งหมายถึงการ load ค่า x ไปยังรีจิสเตอร์ r1 ซึ่งนอกจากการอ้างอิงแบบนี้แล้ว ยังสามารถอ้างอิงที่ address ของ x ดังนี้

```
ld      [2068], %r1
```

นั่นคือทำการ load ค่าที่ address ตำแหน่ง 2068 ไปเก็บไว้ที่รีจิสเตอร์ r1 แทนที่จะอ้างอิงด้วยการใช้ตัวแปร x สามารถใช้ address ได้โดยตรง ซึ่งสามารถคำนวณได้จากตำแหน่งเริ่มต้นของโปรแกรม แล้วเพิ่มไปยังตำแหน่งที่ Instruction นั้นๆ อยู่

ชุดคำสั่งใน ARCTool

โปรแกรมจำลองเวอร์ชั่น 2.1.2 นี้ สนับสนุนชุดคำสั่งต่อไปนี้

```
ADD     AND     ANDN    ADDCC           ANDCC           ANDNCC BA      BN      BNE     BE      BG
BLE     BGE     BL      BGU     BLEU    BCC     BCS     BPOS   BNEG    BVC     BVS     CALL    JMPL
LD      LDSB    LDSH    LDUB    LDUH    OR      ORN     ORCC   ORNCC   RD      RETT    SETHI   ST
STB     STH     SRL     SLL     SRA     SUB     SUBCC   TA     WR      XOR     XNOR    XORCC
XNORCC
```

และสามารถเรียกใช้งาน Synthetic Instruction ได้ดังต่อไปนี้

```
Synthetic Instruction       Instruction Generated                   Comment
not rs1, rd                     xnor rs1, %r0, rd                   1's complement
neg rs2, rd                     sub %r0, rs2, rd                    2's complement
inc rd                          add rd, 1, rd                       increment by 1
dec rd                          sub rd, 1, rd                       decrement by 1
clr rd                          and rd, %r0, rd                     clear a register
cmp rs1, reg_or_imm             subcc rs1, reg_or_imm, %r0          compare, set cc's
tst rs2                         orcc %r0, rs2, %r0                  test
mov reg_or_imm, rd              or %r0, reg_or_imm, rd              move a value
```

ให้ศึกษาจากโปรแกรมต่อไปนี้

```
                .begin
                .org 2048
main:   ld      [x], %r1
                ld      [y], %r8
                ld      [z], %r12
!---------- shift -------------------------
                sll     %r1, 1, %r2
                sll     %r1, 2, %r3
                sll     %r1, 3, %r4
                srl     %r4, 3, %r5
!----------- x or ---------------------
                xor     %r8, 0xFFFFFFFF, %r10
!----------- synthetic ---------------
                inc %r12
                dec %r12
                halt
x:      1
y:      0x00FFFF00
z:      9
```

จากนั้น ให้สังเกตการใช้งาน shift left ซึ่งสามารถนำมาประยุกต์ใช้งานในการคูณ เช่น หาค่าของ 6 * 4 ดังตัวอย่างด้านล่าง

6_{10} = 00000110_2

เมื่อทำการ shift left ไป 2 ครั้ง จะได้

#1 00001100_2

#2 00011000_2 = 24_{10}

นั่นคือ เมื่อต้องการคูณจะเปลี่ยนเป็นการ shift left แทน และเนื่องจากเป็นการทำที่ระดับบิต จึงทำให้เป็นการคูณในรูปของ 2 เท่าไปเรื่อยๆ จากตัวอย่างต้องการคูณด้วย 4 จึงต้องทำการ shift left ไป 2 ครั้ง ($2^2 = 4$) แล้วถ้าเป็นการ shift right ผู้อ่านพอจะเดาได้รึไม่ว่าจะผลลัพธ์จะออกมาอย่างไร และถ้าตัวคูณไม่อยู่ในรูปของ 2 เท่าจะต้องทำอย่างไร ผู้เขียนขอฝากเป็นการบ้านไว้ 2 ข้อนะครับ