การผลิตโค๊ดสำหรับ Procedure Call

Activation Record

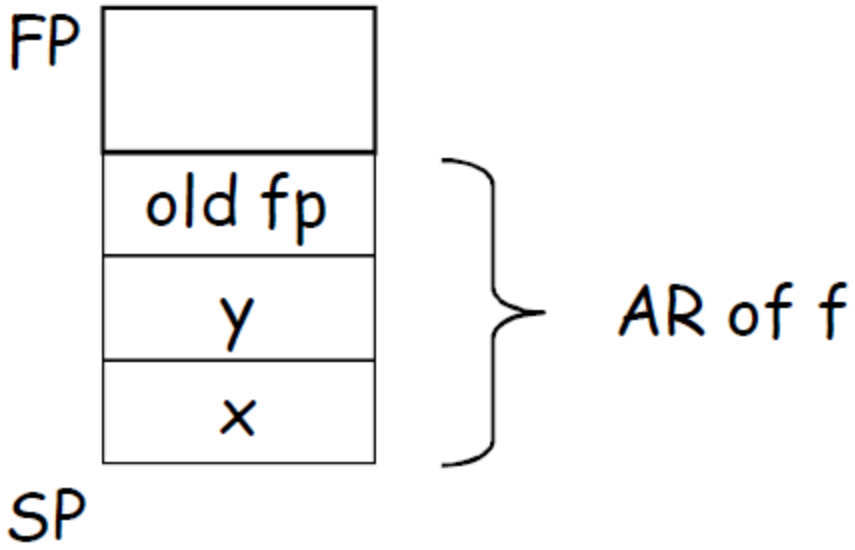
- การผลิตโค๊ดที่เกี่ยวกับ procedure และการ call procedure ขึ้นกับรูปแบบของ AR ที่ออกแบบไว้
- โค๊ดสำหรับ stack machine ที่เราจะผลิตใช้ AR ง่ายๆ
  - ผลลัพธ์อยู่ที่ accumulator เสมอ ไม่จำเป็นต้องเก็บลง AR
  - AR เก็บข้อมูลของ actual parameter ที่ส่งผ่านมาจาก caller
  - สำหรับการเรียก function f(x1, x2, … , xn) เราจะ push x1, …, xn ลงบน stack

Activation Record

- $sp ค่าก่อนหน้าและหลังการเรียก function มีค่าเท่ากัน
  - การผลิตโค๊ดต้องทำตามวินัย stack อย่างเคร่งครัด
- ต้องมี return address
- มี frame pointer ($fp) เอาไว้ใช้ในการอ้างถึงตัวแปรบน stack
- ดังนั้นโดยสรุป AR ของเรามีข้อมูล:
  - Frame pointer
  - Actual parameters
  - Return address

แผนภาพ AR

- พิจารณาการเรียก function f(x, y) จะได้ AR ลักษณะตามด้านล่างนี้



โค๊ดสำหรับ Function Call

- Calling sequence คือกลุ่มคำสั่งที่เกี่ยวข้องกับการเรียกใช้ function ทั้งในส่วน caller และ callee
- คำสั่ง MIPS ที่ใช้ในการนี้: jal label
  - กระโดดไปที่ label และเก็บ address ของคำสั่งถัดไปไว้ที่ $ra
  - ต้องมีคำสั่งเก็บค่าของ $ra ลงใน stack
  - คำสั่ง call ใน X86 ISA จะ push ค่า return address ลง stack อัตโนมัติ

โค๊ดสำหรับ Function Call

- Caller เก็บค่า frame pointer ลงใน stack
- Caller เก็บค่า actual parameter (หรือ argument ที่ส่งผ่านไปยัง callee) ลงบน stack โดยเก็บค่า parameter ที่อยู่ใน order มากที่สุดก่อน ไล่ลงมาจนเก็บค่า parameter ตัวแรกท้ายที่สุด
- Callee เก็บค่าใน $ra (return address) ลงบน stack
- Callee pop ออกจาก stack:
  - Return address
  - ค่า arguments ที่ส่งผ่านมา
  - Frame pointer

โค๊ดสำหรับ Function Call

cgen(f(e_1,…,e_n)) =

```
    sw $fp 0($sp)
    addiu $sp $sp -4
    cgen(e_n)
    sw $a0 0($sp)
    addiu $sp $sp -4
    …
    cgen(e_1)
    sw $a0 0($sp)
    addiu $sp $sp -4
    jal f_entry
```

- นิสิตบอกได้ไหมว่าขณะนี้ขนาดของ stack frame มีค่าเท่าไหร่
  - 4*n + 4

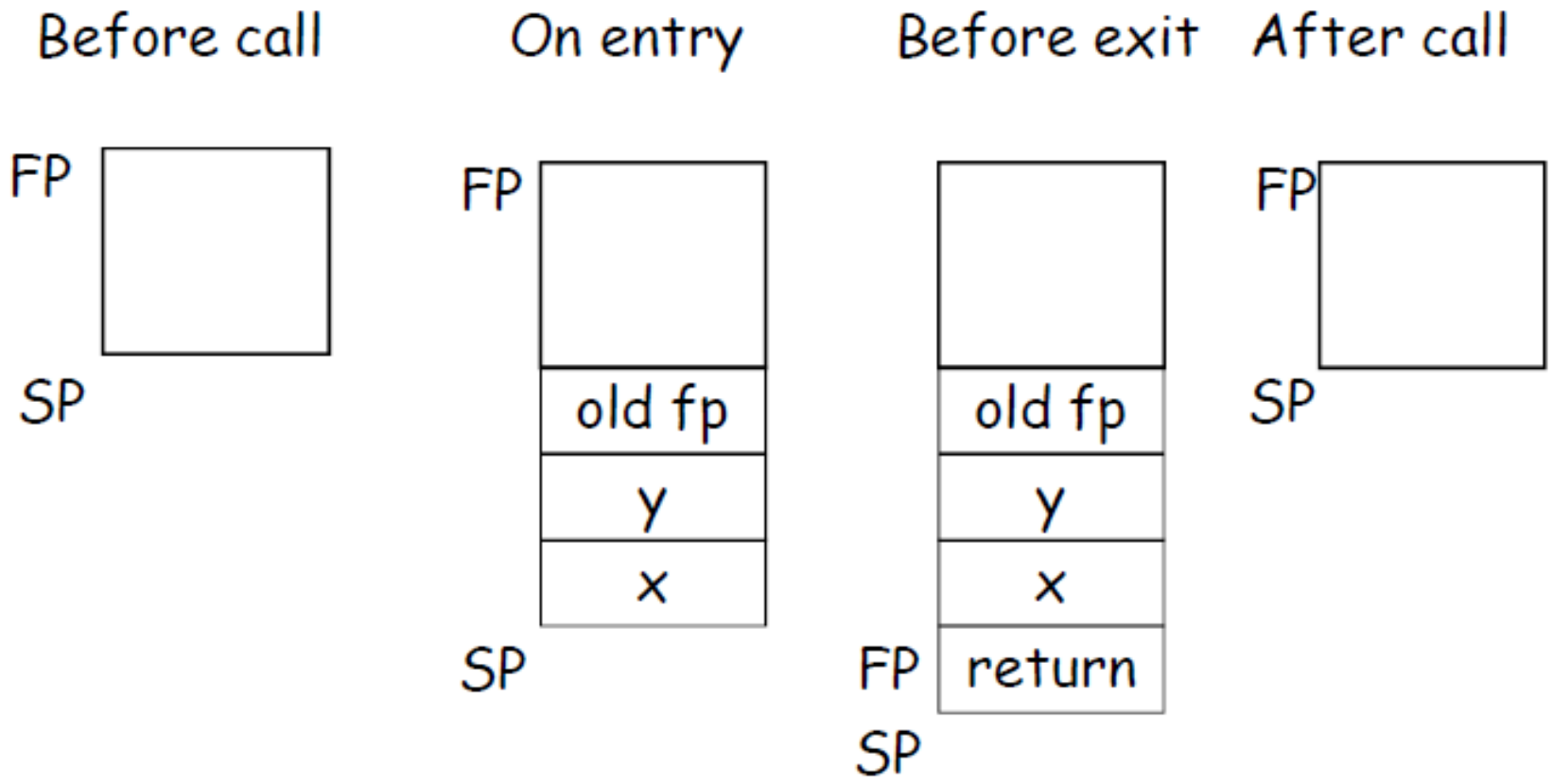
โค๊ดสำหรับ Function Definition

cgen(T f(T1 x1, …, Tn xn) { e }) =

```
move $fp $sp
sw $ra 0($sp)
addiu $sp $sp -4
cgen(e)
lw $ra 4($sp)
addiu $sp $sp z
lw $fp 0($sp)
jr $ra
```

- Callee จะ pop ค่าต่อไปนี้:
  - Return address
  - Actual arguments
  - Frame pointer
- z = 4*n + 8

ตัวอย่าง AR สำหรับ f(x, y)



โค๊ดสำหรับตัวแปร

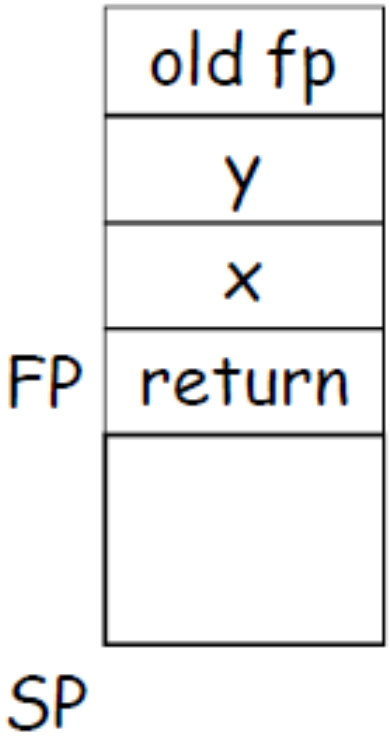
- สำหรับ stack machine ที่เราได้กล่าวมา ตัวแปรจริงๆที่ต้องพิจารณาคือ actual arguments (หรือ parameters ของ function)
  - ตัวแปรทุกๆตัวอยู่ใน AR
  - ถูก push โดย caller
- เราจะอ้างไปถึงตัวแปรเหล่านี้ได้อย่างไร ใช้ $sp?
  - ทำได้ลำบากเพราะ AR สำหรับแต่ละ function เติบโตไม่เท่ากัน
  - เราไม่สามารถอ้างไปถึงตัวแปรเหล่านั้นได้โดย offset ที่ตายตัว

Frame Pointer กับการอ้างถึงตัวแปร

- การอ้างไปถึงตัวแปรจะสะดวกกว่ามากถ้าใช้ frame pointer
  - $fp ที่เก็บค่า frame pointer จะชี้ไปที่ตำแหน่ง return address บน stack เสมอ
  - $fp สำหรับแต่ละ AR จะไม่มีการเปลี่ยนตำแหน่ง
  - ดังนั้นสามารถนำไปใช้อ้างถึงตัวแปรได้โดยใช้ offset ที่ตายตัว
- ให้ xi แทน parameter ตัวที่ i (= 1, ..., n) การอ้างไปถึง xi จะใช้ assembly code ต่อไปนี้:

lw $a0, z($fp) โดย z = 4 * i

ตัวอย่าง T f(Tx x, Ty y) { e }



- ตัวแปร x จะถูกชี้ด้วย $fp + 4
- ตัวแปร y จะถูกชี้ด้วย $fp + 8

สรุปโดยรวม

- การออกแบบ AR กับการผลิต assembly code จะต้องทำควบคู่กันและเป็นไปในทิศทางเดียวกัน
- การผลิตโค๊ดสำหรับ stack machine ในลักษณะที่เรากล่าวถึงสามารถกระทำโดยใช้การ traverse AST ในลักษณะ top-down
- คอมไพเลอร์ที่ใช้งานจริงปรับปรุงการผลิตโค๊ดให้ดีขึ้นเช่น
  - เก็บค่าตัวแปรลง register แทนที่จะเก็บลง stack
  - ค่าผลลัพธ์ระหว่างกลางมีตำแหน่งที่กำหนดไว้ใน AR โดยตรง ไม่ต้อง push และ pop จาก stack อยู่ตลอด

Example

```
int sumto(int x) { if (x == 0 ) return 0; else return x + sumto(x-1); }
```

```
sumto:
        move    $fp, $sp
        sw      $ra, 0($sp)
        addi    $sp, $sp, -4
        lw      $a0, 4($fp)
        sw      $a0, 0($sp)
        addi    $sp, $sp, -4
        li      $a0, 0
        lw      $t1, 4($sp)
        addi    $sp, $sp, 4
        beq     $a0, $t1 true1
false1:
        lw      $a0, 4($fp)
        sw      $a0, 0($sp)
        addi    $sp, $sp, -4
        sw      $fp, 0($sp)
        addi    $sp, $sp, -4
        lw      $a0, 4($fp)
        sw      $a0, 0($sp)
        addi    $sp, $sp, -4
        li      $a0, 1
        lw      $t1, 4($sp)
        sub     $a0, $t1, $a0
        addi    $sp, $sp, 4
        sw      $a0, 0($sp)
        addi    $sp, $sp, -4
        jal     sumto
        lw      $t1, 4($sp)
        add     $a0, $t1, $a0
        addi    $sp, $sp, 4
        j       endif
true1:
        li      $a0, 0
endif:
        lw      $ra, 4($sp)
        addi    $sp, $sp, 12
        lw      $fp, 0($sp)
        jr      $ra
```