Tutorial-7: The Stack

AbdulWahab Kabani

Outline

- The Stack – Introduction
- Push Instruction
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

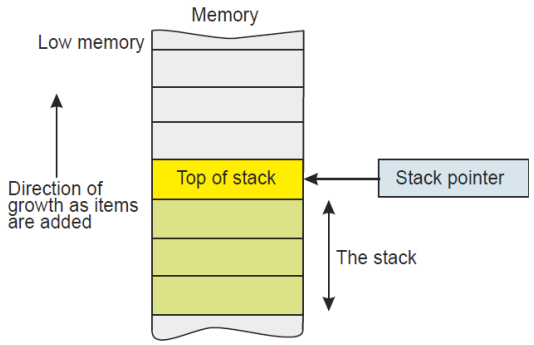
Outline

- **The Stack – Introduction**
- Push Instruction
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

The Stack - Introduction

- Last-in-First-out
- Many applications:
  - Storing subroutine return address
  - Passing parameters from a program to a subroutine
  - Providing temporary storage (local workspace) in a subroutine.

The Stack - Introduction



5

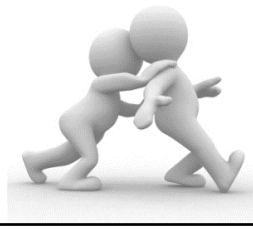
Outline

- **The Stack – Introduction**
- **Push Instruction**
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

6

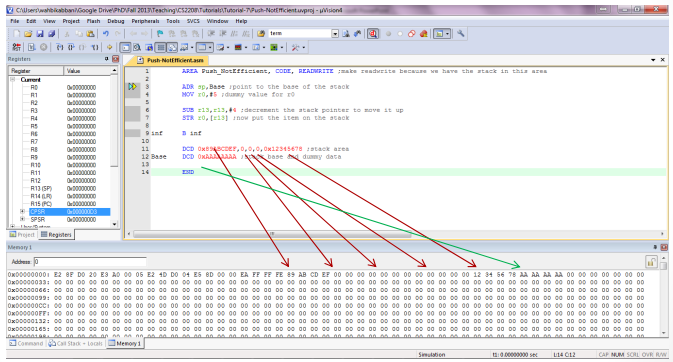
Push Instruction

- What is the register that is used as a stack pointer?
  - R13 or (SP)
- How can we **push** an item in register r0 to the stack?
  - Hint: update r13 and store the item in r0 in memory
    - SUB **r13**,r13,#4
    - STR r0,**[r13]**



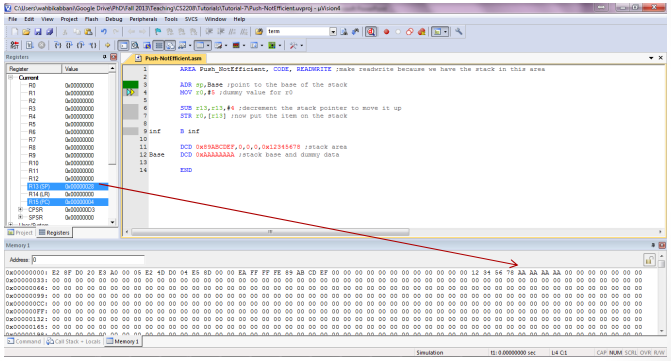
7

Push Instruction



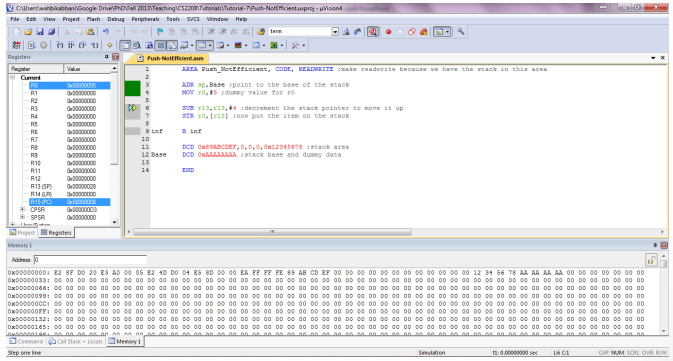
8

Push Instruction



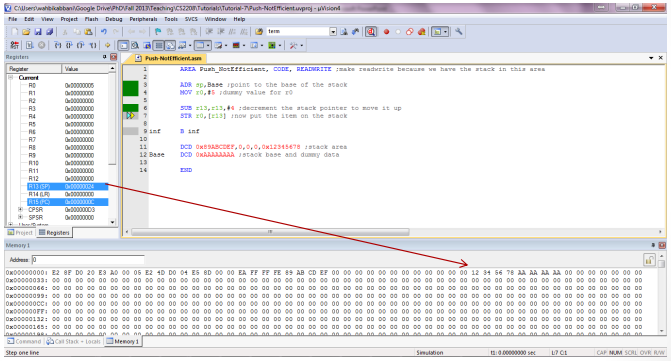
9

Push Instruction



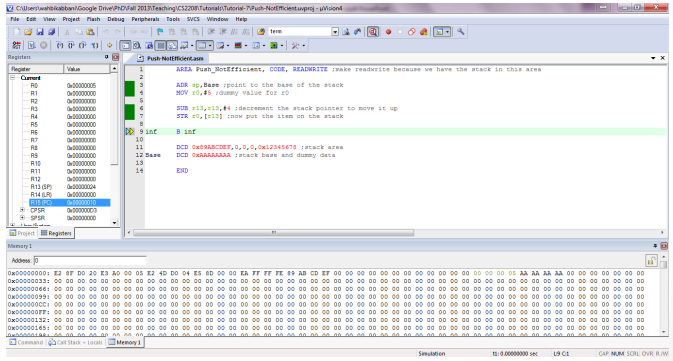
10

Push Instruction



11

Push Instruction



12

Outline

- **The Stack – Introduction**
- **Push Instruction**
  - **Push Instruction (Efficient)**
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

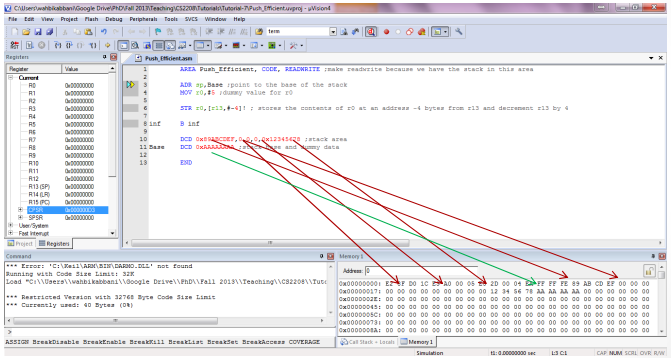
13

Push Instruction (Efficient)

- Efficient way:
  - STR **r0**,[r13,#-4]!
  - Pre-update Instruction
  - stores the contents of r0 at an address -4 bytes from r13 (4 bytes above it)
  - Decrement the contents of r13 by 4
- Be efficient in your assignments!

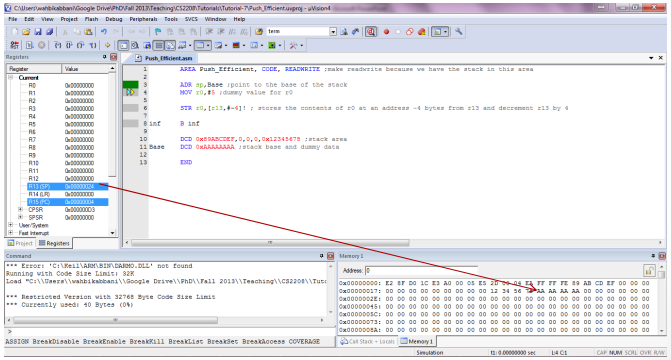
14

Push Instruction (Efficient)



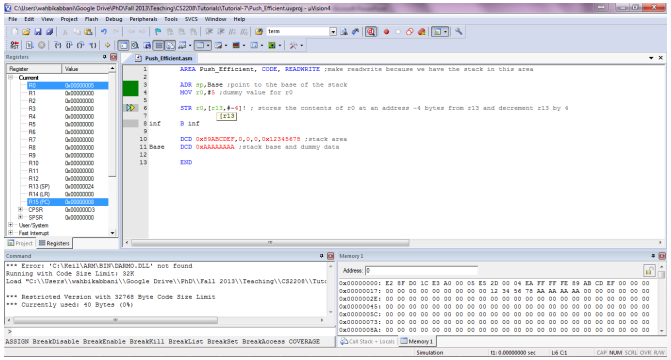
15

Push Instruction (Efficient)



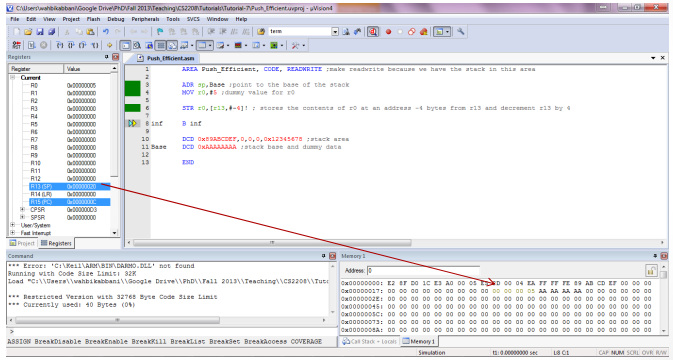
16

Push Instruction (Efficient)



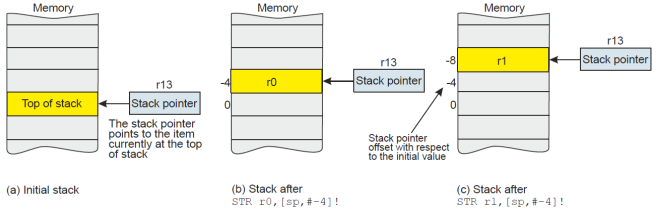
17

Push Instruction (Efficient)



18

Push Instruction (Efficient)



19

Outline

- The Stack – Introduction
- Push Instruction
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

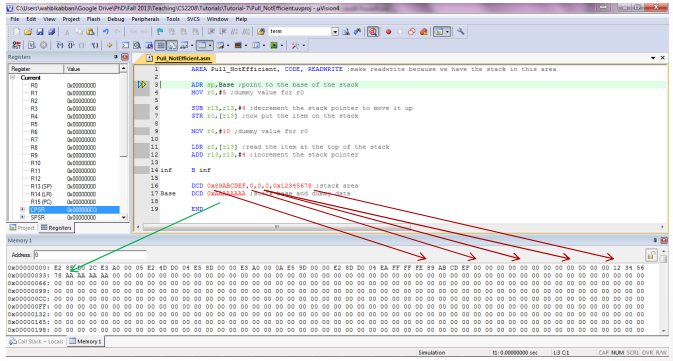
20

Pull Instruction

- How can we pull (pop) a word off the stack?
  - **Hint**: read the item currently at the top and increment stack pointer
  - Answer:
    - LDR **r0**,[r13]
    - ADD **r13**,r13,#4

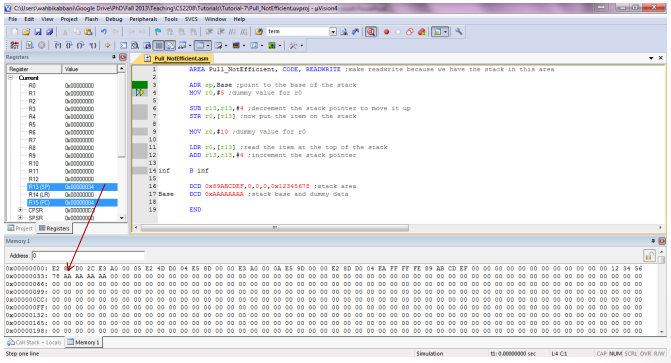
21

Pull Instruction



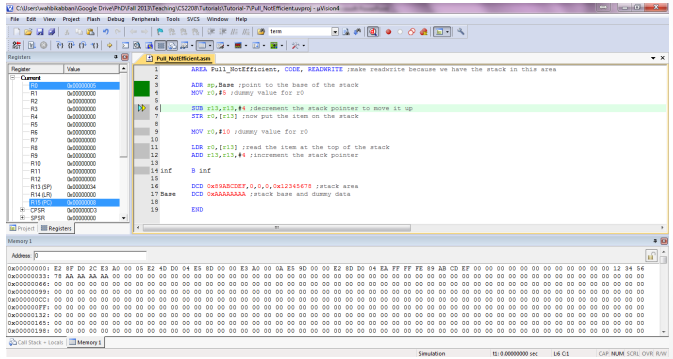
22

Pull Instruction



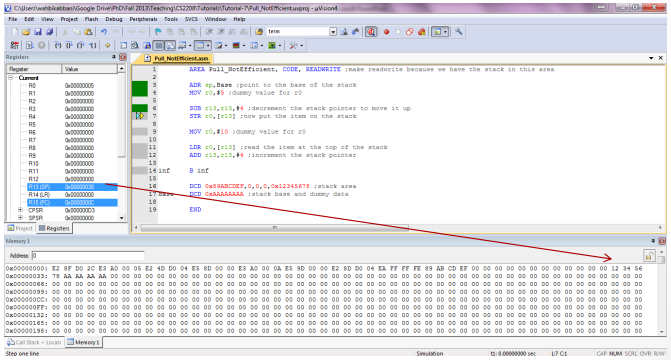
23

Pull Instruction



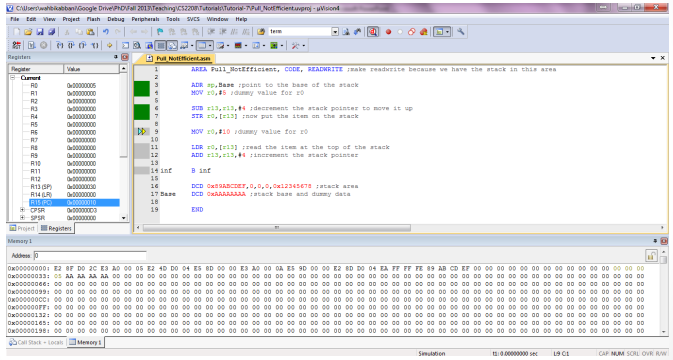
24

Pull Instruction



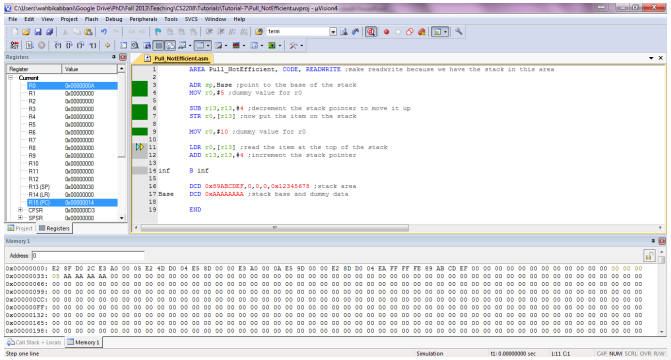
25

Pull Instruction



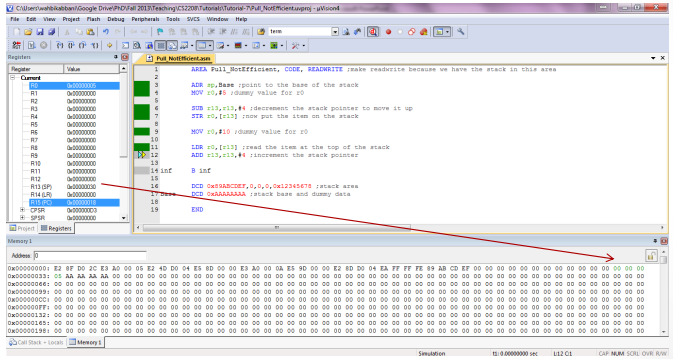
26

Pull Instruction



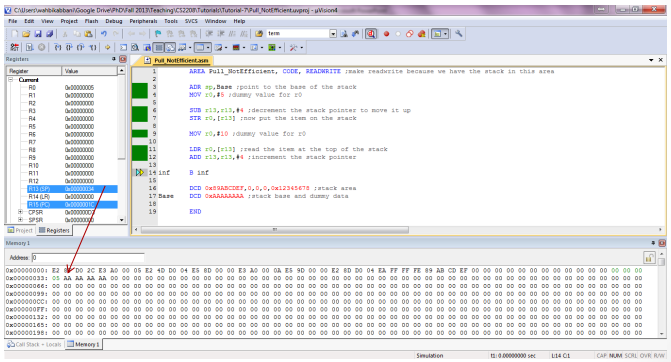
27

Pull Instruction



28

Pull Instruction



29

Outline

- The Stack – Introduction
- Push Instruction
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

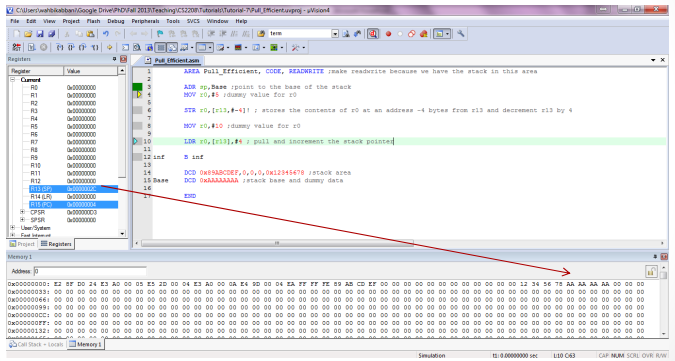
30

Pull Instruction (Efficient)

- Efficient way:
  - LDR **r0**,[r13],#4
- Post-update Instruction
- Again, make sure to use the efficient way in your assignments

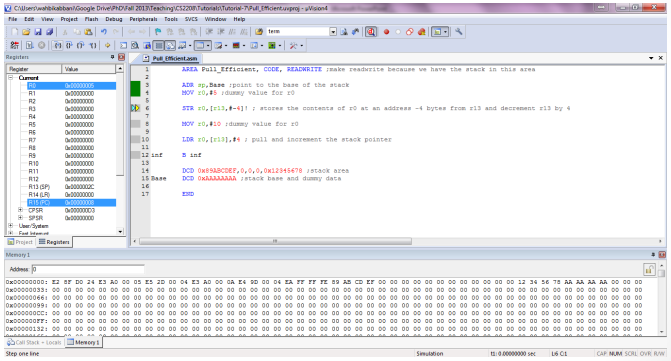
31

Pull Instruction (Efficient)



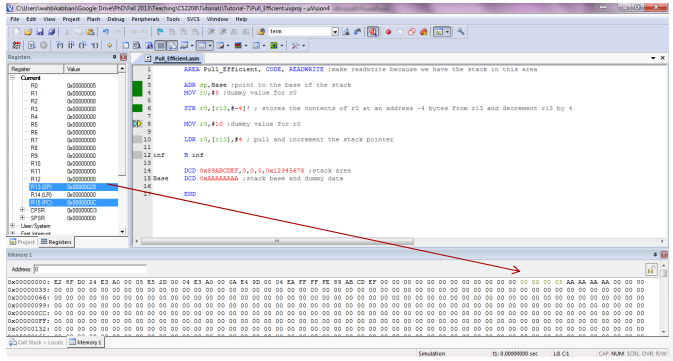
32

Pull Instruction (Efficient)



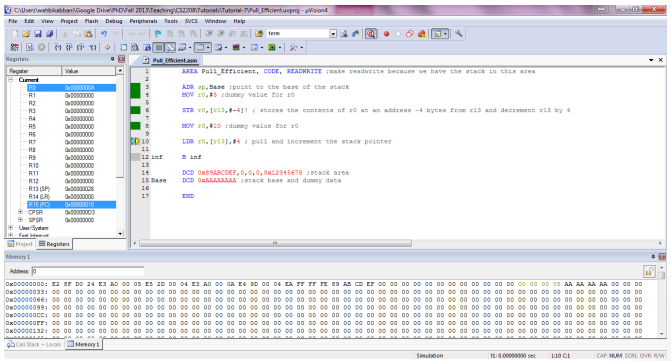
33

Pull Instruction (Efficient)



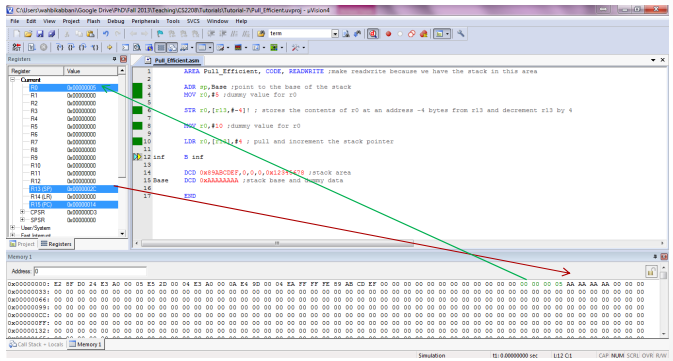
34

Pull Instruction (Efficient)



35

Pull Instruction (Efficient)



36

Push/Pull Comparison

Push	Pull (Pop)
STR **r0**,[r13,#-4]!	LDR **r0**,[r13],#4
stores the contents of r0 at an address -4	read the item currently at top
Decrement the contents of r13 by 4	increment stack pointer

37

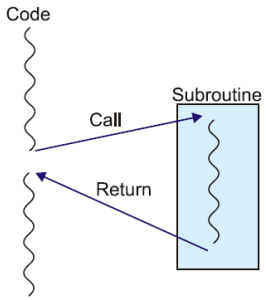
Outline

- **The Stack – Introduction**
- **Push Instruction**
  - **Push Instruction (Efficient)**
- **Pull Instruction**
  - **Pull Instruction (Efficient)**
- **Subroutine**
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

38

Subroutine

- A subroutine is a piece of code that is called, executed and a return is made to the calling point.
- Implement a *"function" in high level languages*
- Instruction: BL (branch to subroutine)



39

Subroutine

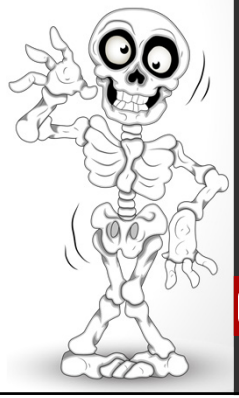
- BL ABC (branch to subroutine)
  - saves a copy of the return address in the link register (r14)
  - A return back to the calling point is made by copying the return address from the link register to the program counter, r15.

40

Subroutine

- The skeleton of a typical processor call and return routine:
  - Main Code
    - BL XYZ → call XYZ
    - … ← Return here
  - XYZ ← Subroutine
    - …
    - MOV **pc**,lr

copy saved address to PC to return



41

Subroutine

- a simple subroutine called SQR1
- SQR1 calculates $x^2 + 1$
  - x is in register r0
  - Result will be returned in r0
- **Question**: How can we call this subroutine?
  - BL SQR1

42

Subroutine

- Help me write the code. We will do it step by step:
  - Initialize register r0 with the value: 4
    - MOV **r0**,#4
  - Call subroutine SQR1:
    - BL **SQR1**
  - First line of subroutine SQR1, multiply x by x
    - SQR1 MUL **r1**,r0,r0
  - Add 1 to get $x^2 + 1$. Put the result in r0
    - ADD **r0**,r1,#1
  - Return
    - MOV **pc**,lr

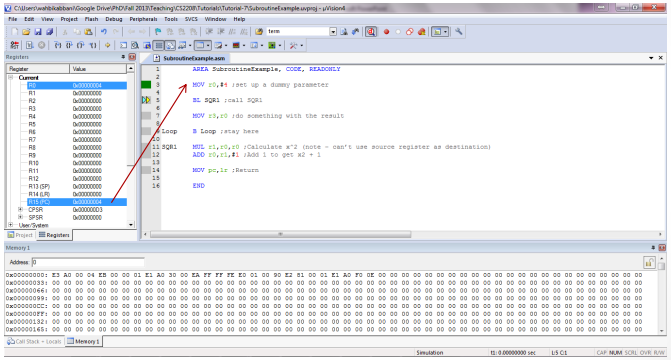
43

Subroutine

- Thank you for helping me write the code.
- Now, let's run the program

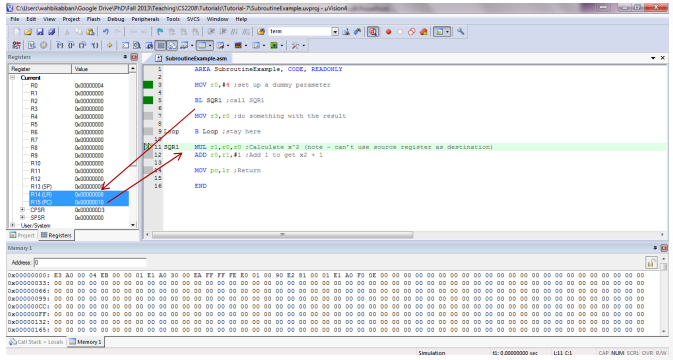
44

Subroutine



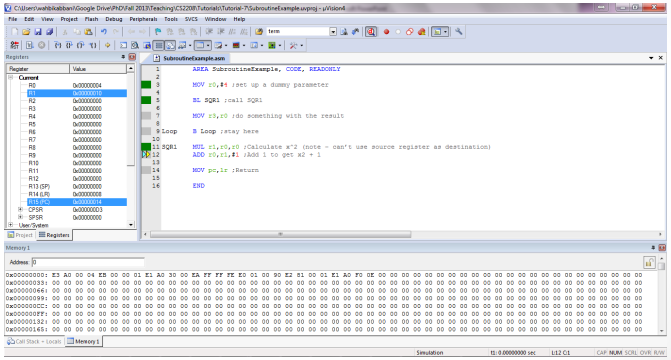
45

Subroutine



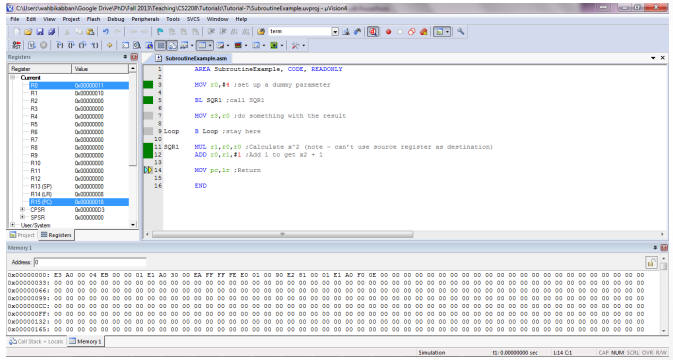
46

Subroutine



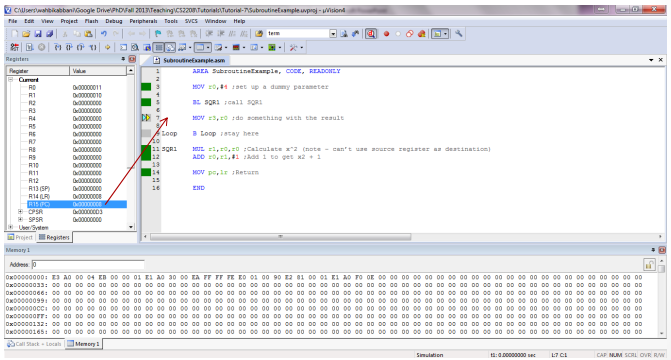
47

Subroutine



48

Subroutine



49

Outline

- The Stack – Introduction
- Push Instruction
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

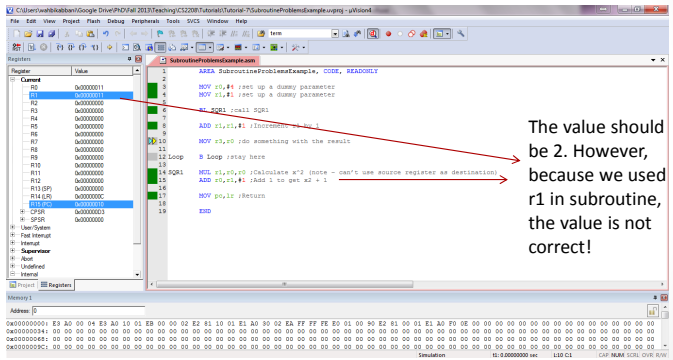
50

Subroutine – Problems

- Our code is NOT perfect
- We will modify the code to explore a possible problem

51

Subroutine – Problems



52

Subroutine – Problems

- Two problems:
  - Problem-1:
    - We used r1 to return the result of x^2
    - What happens to the original value stored in this register?
  - Problem-2:
    - subroutine can't call another subroutine
    - the return address is in lr will be overwritten

53

Subroutine – Problems

- Two Steps to solve both problems:
  - Save the content of the link register and other needed registers in stack
  - Why is the stack the best place to save these values?
    - Values won't be overwritten

54

Subroutine – Problems

- Remember our push/pull instructions:
  - Push: STR **r0**,[r13,#-4]!
  - Pull : LDR **r0**,[r13],#4
- Using these instructions, help me modify the code of the subroutine.
  - Save link register on the stack
    - SQR1 STR lr **,[sp,#-4]!**
  - We will be overwritting r1. What is the instruction to save it in the stack?
    - STR r1 **,[sp,#-4]!**

55

Subroutine – Problems

- Remember our push/pull instructions:
  - Push: STR **r0**,[r13,#-4]!
  - Pull : LDR **r0**,[r13],#4
- Using these instructions, help me modify the code of the subroutine.
  - At the end of the subroutine, we want to restore the original value of r1:
    - LDR **r1** ,[sp] ,#4
  - At the end of the subroutine, restore the value of the link register:
    - LDR **lr** ,[sp] ,#4

56

Subroutine – Problems

- Let's put everything together:

SQR1 STR lr ,[sp,#-4]! ;Save link register on the stack

STR r1 ,[sp,#-4]! ;Save register r1 on the stack

MUL r1,r0,r0 ;Calculate x^2

ADD r0,r1,#1 ;Add 1 to get $x^2 + 1$

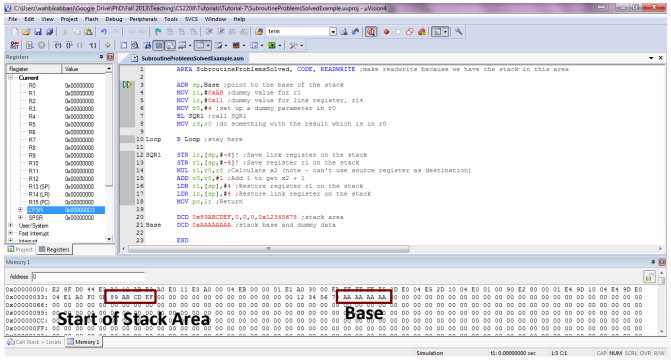
LDR r1 ,[sp] ,#4 ;Restore register r1 from the stack

LDR lr ,[sp] ,#4 ;Restore link register from the stack

MOV pc,lr ;Return

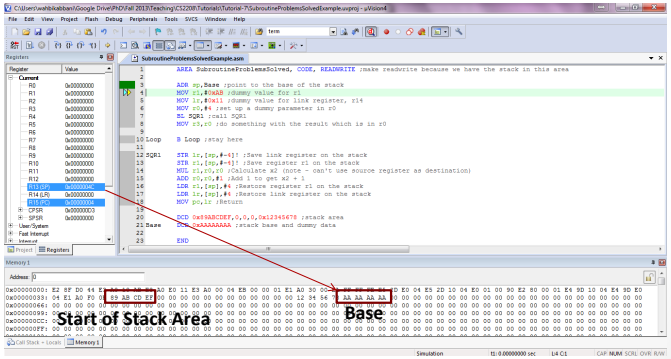
57

Subroutine – Problems



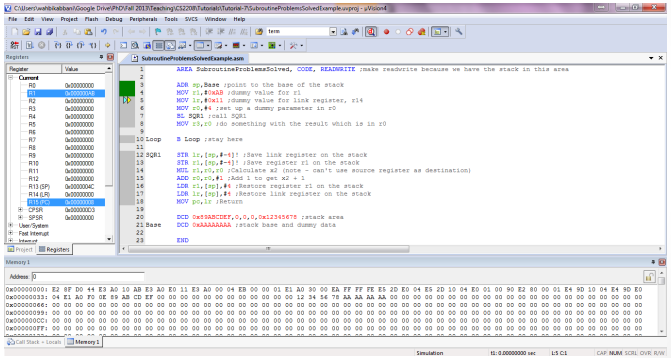
58

Subroutine – Problems



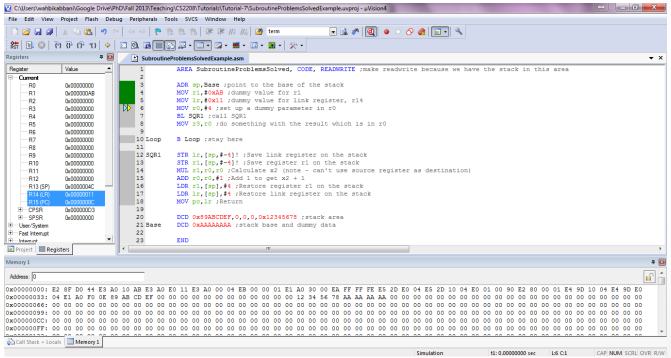
59

Subroutine – Problems



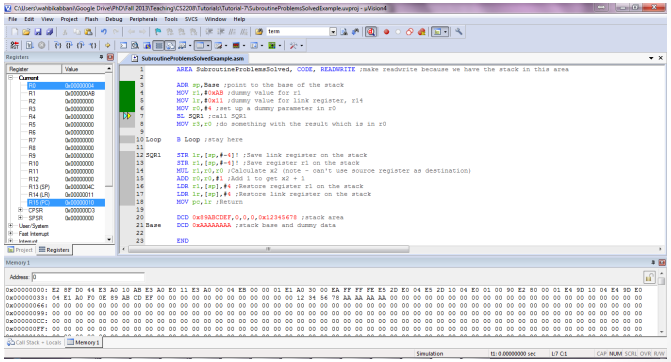
60

Subroutine – Problems



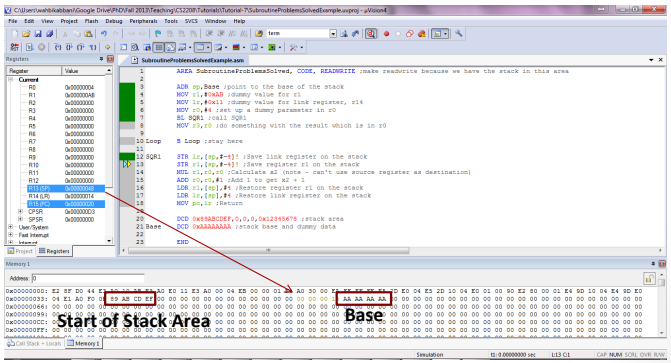
61

Subroutine – Problems



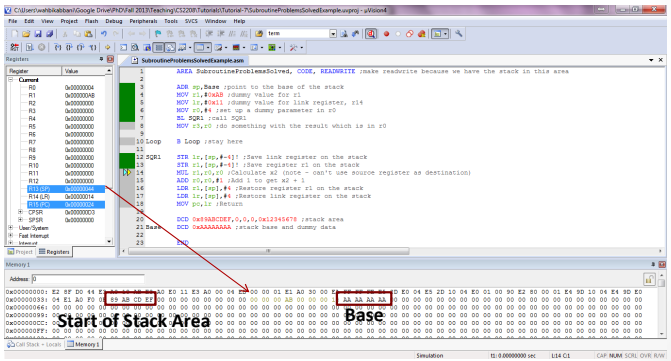
62

Subroutine – Problems



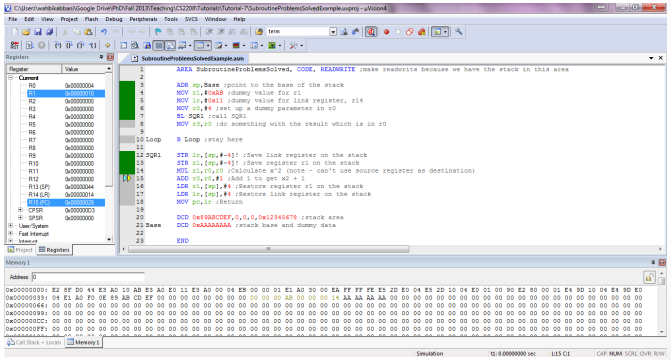
63

Subroutine – Problems



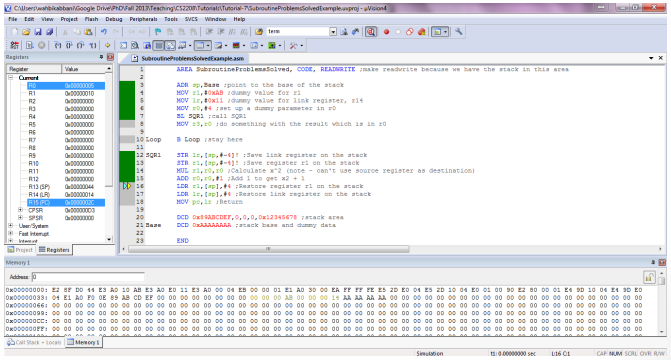
64

Subroutine – Problems



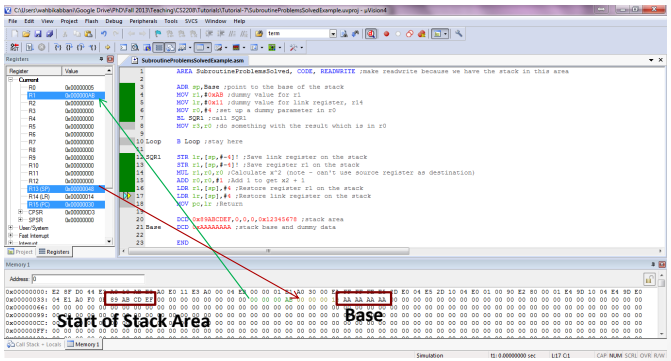
65

Subroutine – Problems



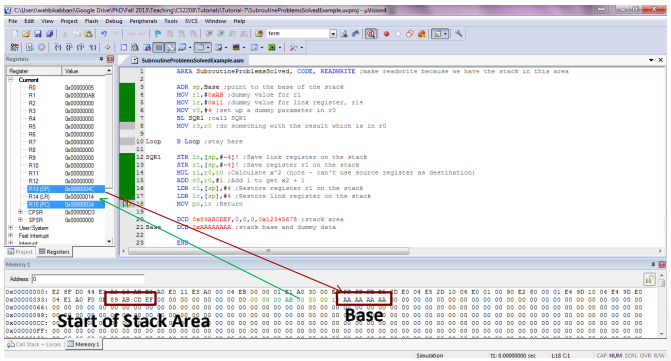
66

Subroutine – Problems



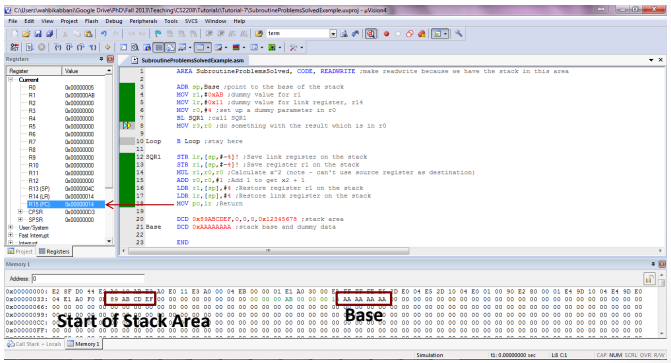
67

Subroutine – Problems



68

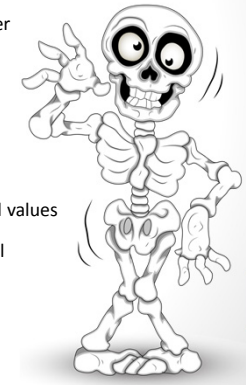
Subroutine – Problems



69

Subroutine – Problems

- The skeleton of a typical pushing/pulling to a stack in a subroutine:
  - STR lr **,[sp,#-4]!** → Save the link register
  - STR r1 **,[sp,#-4]!** → Save the registers
  - ….
  - LDR **r1** ,[sp] ,#4 → Restore the original values
  - LDR **lr** ,[sp] ,#4 → Restore the original Link register value
  - MOV **pc**,lr → Return



70

Outline

- The Stack – Introduction
- Push Instruction
  - Push Instruction (Efficient)
- Pull Instruction
  - Pull Instruction (Efficient)
- Subroutine
  - Subroutine – Problems
  - Subroutine – Efficient push/pull registers

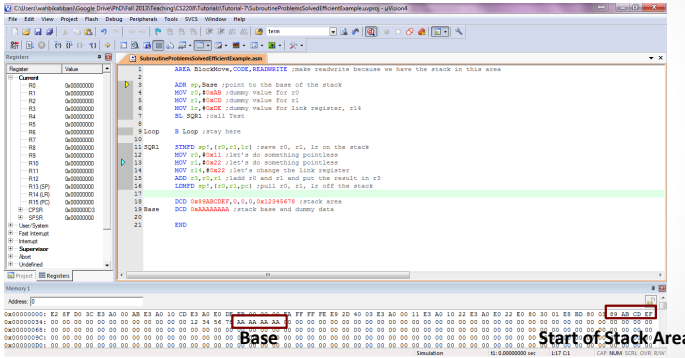
71

Subroutine – Efficient push/pull registers

- The previous code solved the 2 problems we had
- However, there's a more efficient way to save/restore multiple registers in/from memory
  - STMFD ;Push a group of registers on the stack
  - LDMFD ;Pull a group of registers off the stack

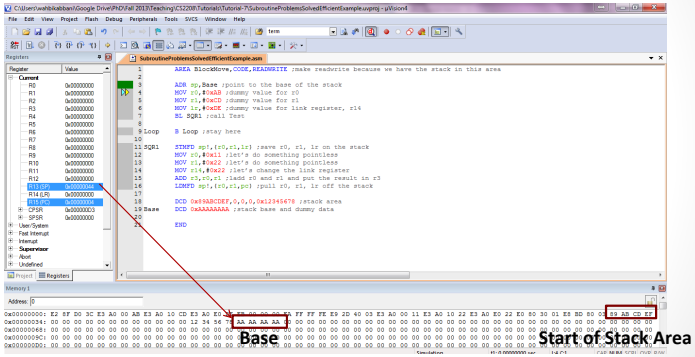
72

Subroutine – Efficient push/pull registers



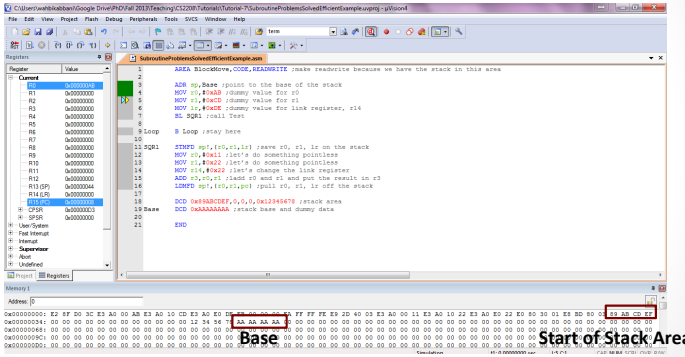
73

Subroutine – Efficient push/pull registers



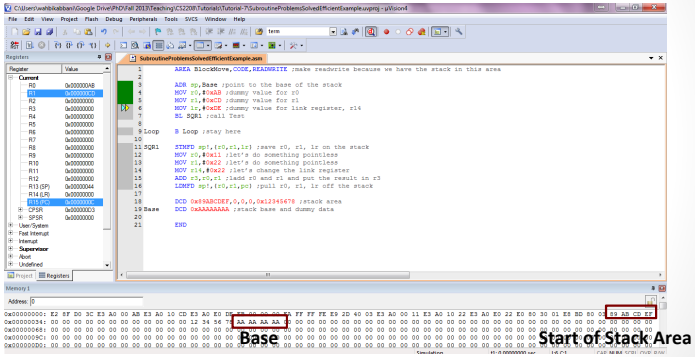
74

Subroutine – Efficient push/pull registers



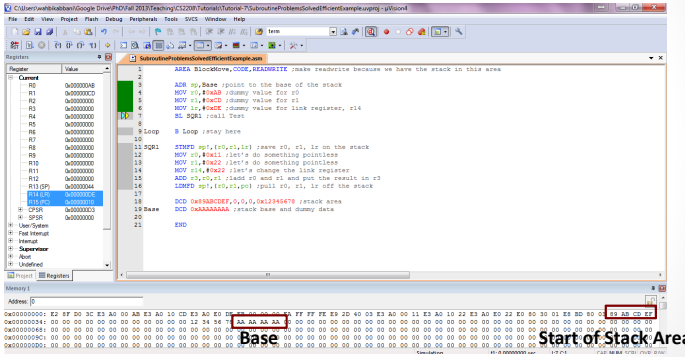
75

Subroutine – Efficient push/pull registers



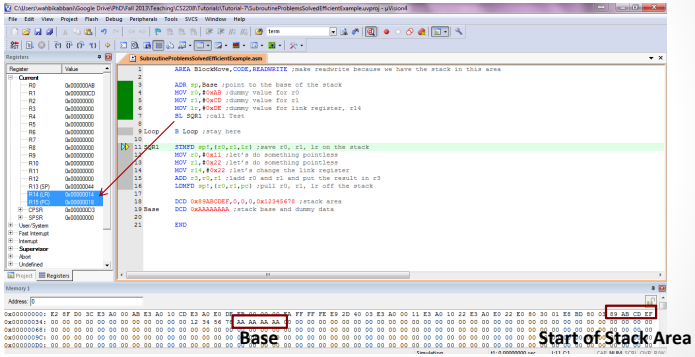
76

Subroutine – Efficient push/pull registers



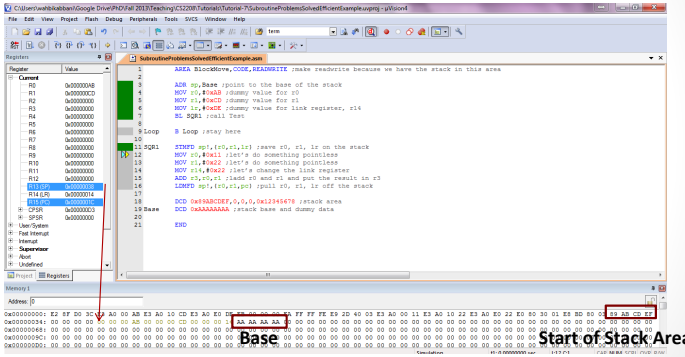
77

Subroutine – Efficient push/pull registers



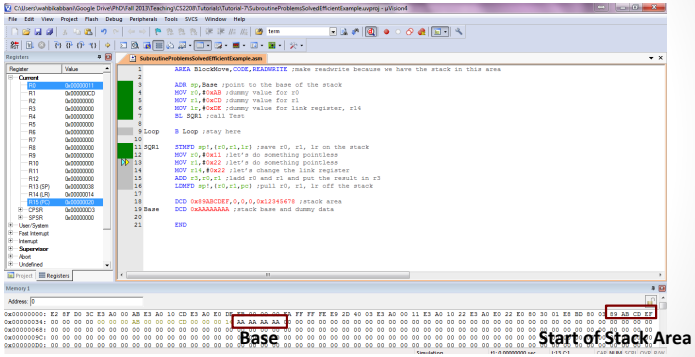
78

Subroutine – Efficient push/pull registers



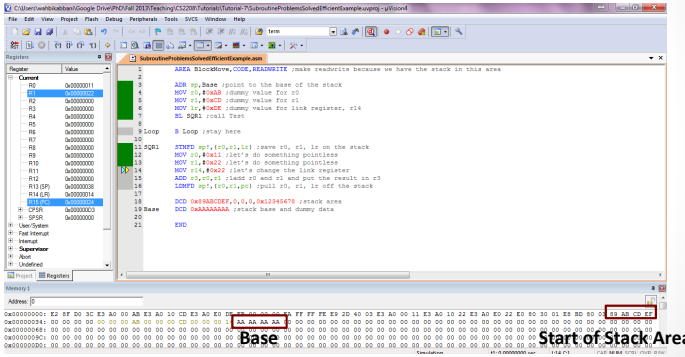
79

Subroutine – Efficient push/pull registers



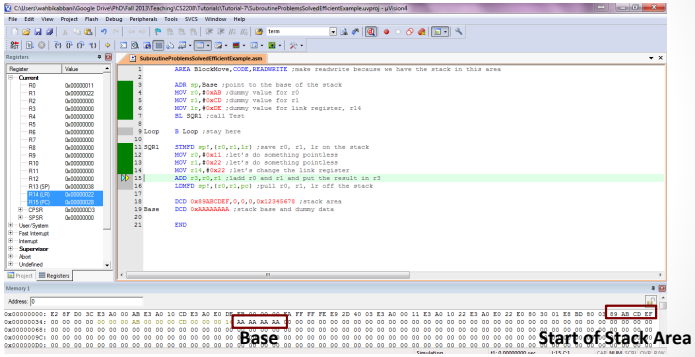
80

Subroutine – Efficient push/pull registers



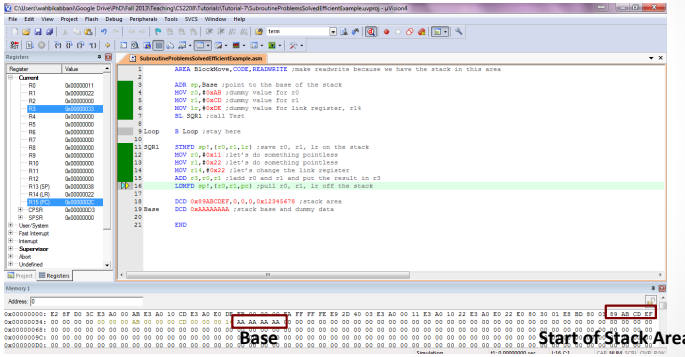
81

Subroutine – Efficient push/pull registers



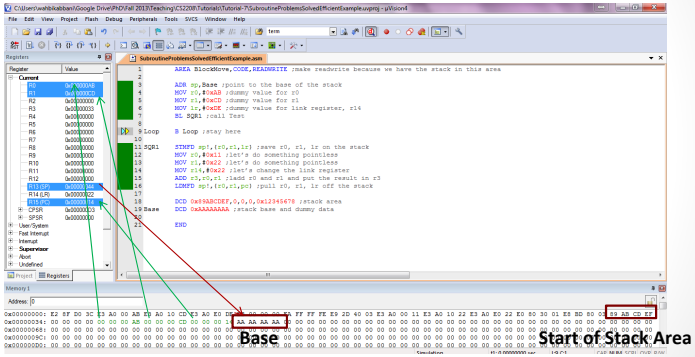
82

Subroutine – Efficient push/pull registers



83

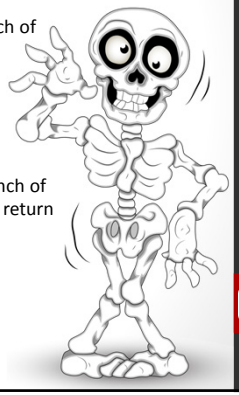
Subroutine – Efficient push/pull registers



84

Subroutine – Efficient push/pull registers

- The skeleton of a typical **Efficient** pushing/pulling to a stack in a subroutine (call/return):
  - STMFD sp!,{r0,r1,lr} → Save a bunch of registers in memory
  - ….
  - LDMFD sp!,{r0,r1,pc} → Restore a bunch of registers and return



85

Outline

86

Summary

Push	Pull (Pop)
STR **r0**,[r13,#-4]!	LDR **r0**,[r13],#4
stores the contents of r0 at an address -4	read the item currently at top
Decrement the contents of r13 by 4	increment stack pointer

87

Summary

- The skeleton of a typical processor call and return routine:
  - Main Code
    - BL XYZ → call XYZ
    - … ← Return here
  - XYZ ← Subroutine
    - …
    - MOV **pc**,lr → copy saved address to PC to return

88

Summary

- The skeleton of a typical pushing/pulling to a stack in a subroutine:
  - STR lr **,[sp,#-4]!** → Save the link register
  - STR r1 **,[sp,#-4]!** → Save the registers
  - **….**
  - LDR **r1** ,[sp] ,#4 → Restore the original values
  - LDR **lr** ,[sp] ,#4 → Restore the original Link register value
  - MOV **pc,**lr → Return

Summary

- The skeleton of a typical **Efficient** pushing/pulling to a stack in a subroutine (call/return):
  - STMFD sp!,{r0,r1,lr} → Save a bunch of registers in memory
  - **….**
  - LDMFD sp!,{r0,r1,pc} → Restore a bunch of registers and return