

Tutorial-7: The Stack

AbdulWahab Kabani

[1]

Outline

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

[2]

Outline

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

[3]

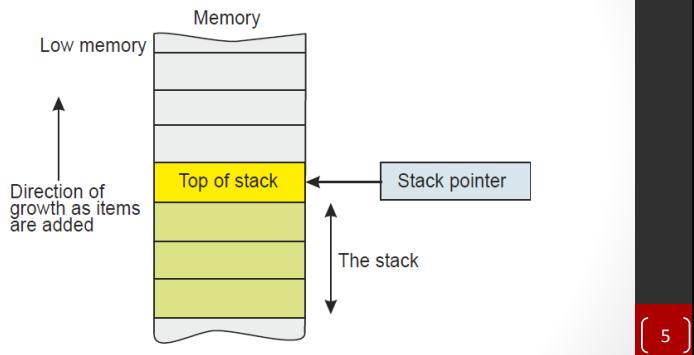
The Stack - Introduction

- 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.



[4]

The Stack - Introduction



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

The screenshot shows the QEMU debugger interface with the assembly window displaying the following code:

```

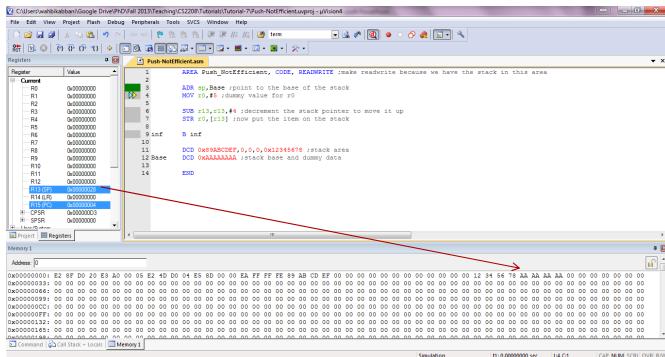
1 ADDA_Push_Notification, CODE, READWRITE :make readable because we have the stack in this area
2
3 ADD R15,SP,-40             ;Base point to the base of the stack
4 MOV R15,R14                 ;dummy value for r0
5
6 SCD R15,L12+4              ;decrement the stack pointer to move it up
7 L12:                         ;L12: now put the item on the stack
8
9 B inf
10
11 ADDA_Push_Notification, G_O, READONLY :reserve areas
12 Base DCD 00000000           ;these are dummy data
13
14 END

```

The memory dump window shows the stack starting at address 0x00000000, with the first 40 bytes filled with zeros. Red arrows point from the assembly labels to the corresponding memory locations, indicating the state of memory after the push operation.

[8]

Push Instruction



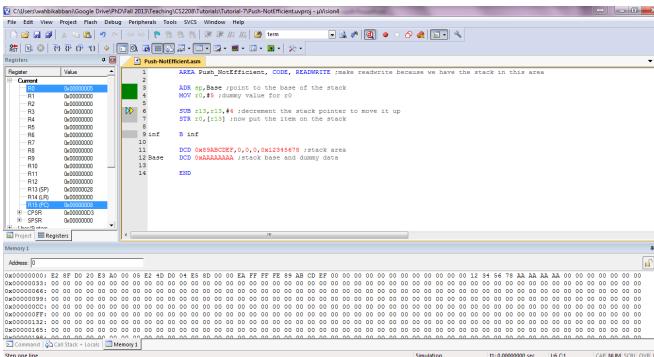
```

AREA Push_NotEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD r0,sp, #8           ;point to the base of the stack
    MOV r0,$8                ;dummy value for r0
    SUB r0,r1, #4            ;decrement the stack pointer to move it up
    STR r0,[r1]               ;now put the item on the stack
    B inf
    DCD 0x9999CCE7,0,0,0x012345678 ;stack area
    DCD 0A000000000000000000000000000000 ;stack base and dummy data
    B end

```

[9]

Push Instruction



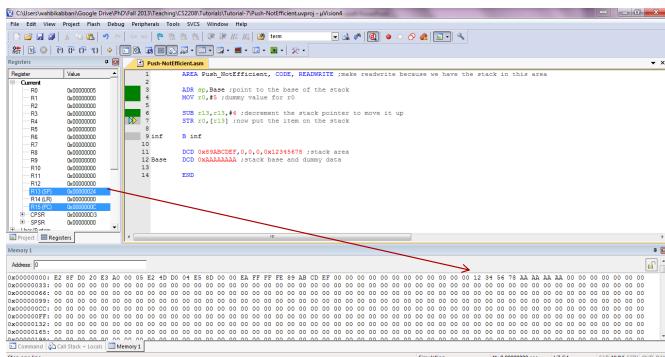
```

AREA Push_NotEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD r0,sp, #8           ;point to the base of the stack
    MOV r0,$8                ;dummy value for r0
    SUB r0,r1, #4            ;decrement the stack pointer to move it up
    STR r0,[r1]               ;now put the item on the stack
    B inf
    DCD 0x9999CCE7,0,0,0x012345678 ;stack area
    DCD 0A000000000000000000000000000000 ;stack base and dummy data
    B end

```

[10]

Push Instruction



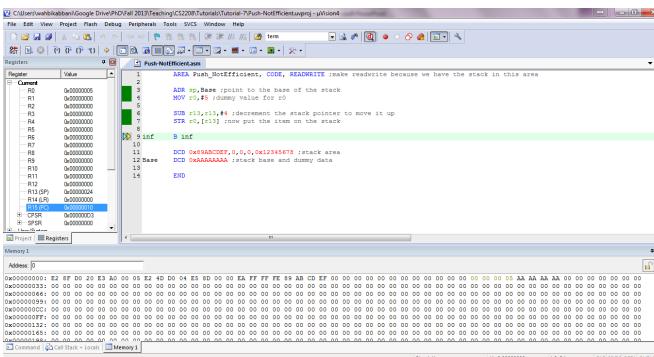
```

AREA Push_NotEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD r0,sp, #8           ;point to the base of the stack
    MOV r0,$8                ;dummy value for r0
    SUB r0,r1, #4            ;decrement the stack pointer to move it up
    STR r0,[r1]               ;now put the item on the stack
    B inf
    DCD 0x9999CCE7,0,0,0x012345678 ;stack area
    DCD 0A000000000000000000000000000000 ;stack base and dummy data
    B end

```

[11]

Push Instruction



```

AREA Push_NotEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD r0,sp, #8           ;point to the base of the stack
    MOV r0,$8                ;dummy value for r0
    SUB r0,r1, #4            ;decrement the stack pointer to move it up
    STR r0,[r1]               ;now put the item on the stack
    B inf
    DCD 0x9999CCE7,0,0,0x012345678 ;stack area
    DCD 0A000000000000000000000000000000 ;stack base and dummy data
    B end

```

[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)

The screenshot shows the Win32 API debugger interface. The assembly window displays the following code:

```
ARM Push_Efficient_CODE, READDISTINCT ; make read/write because we have the stack in this area
    1 ADR r0,[r1] ; point to the base of the stack
    2 ADD r0,r0, #8 ; dummy value for r0
    3 NOT r0,r0, #1 ; stores the contents of r0 at an address -4 bytes from r1 and decrement r1 by 4
    4 STR r0,[r1,#-4]!
    5 B inf
    6
    7 DCD 0xaaaaaaaa,0,0,0x12345678 ;stack area
    8 DCD 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaa ;stack base and dummy data
    9
    10 END
```

The registers window shows:

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14_LR	0x00000002
R15	0x00000000
PSR	0x40000000
SP	0x00000000
LR	0x00000000
PC	0x00000000

The stack window shows:

Address	Value
0x00000000	FF FF E9 80 00 00 00 00 00 00 00 00 00 00 00 00
0x00000001	E2 EF 00 IC E0 A0 00 00 00 00 00 00 00 00 00 00 00
0x00000002	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000003	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000004	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000005	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000006	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000007	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000008	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000009	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

The memory dump window shows:

Address	Value
0x00000000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000001	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000002	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000003	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000004	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000005	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000006	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000007	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000008	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00000009	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

The coverage analysis window shows:

Address	Coverage
0x00000000	0%
0x00000001	0%
0x00000002	0%
0x00000003	0%
0x00000004	0%
0x00000005	0%
0x00000006	0%
0x00000007	0%
0x00000008	0%
0x00000009	0%
0x0000000A	0%
0x0000000B	0%
0x0000000C	0%
0x0000000D	0%
0x0000000E	0%
0x0000000F	0%

[16]

Push Instruction (Efficient)

```

1 AREA Push_Efficient, CODE, READWRITE ;make readable because we have the stack in this area
2
3 ADD r0,Base ;point to the base of the stack
4 NOT r0,#$ ;dummy value for r0
5
6 STR r0,[r13,-4]! ; stores the contents of r0 at an address -4 bytes from r13 and decrement r13 by 4
7
8 Ldf
9
10 DCD 0x433ABCDEF,0,0,0x12345678 ;stack area
11 Base DCD 0xAAAAAAA ;stack base and dummy data
12
13 END

```

[17]

Push Instruction (Efficient)

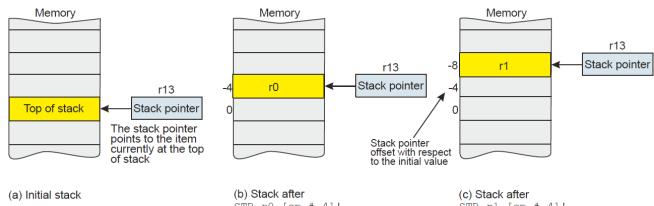
```

1 AREA Push_Efficient, CODE, READWRITE ;make readable because we have the stack in this area
2
3 ADD r0,Base ;point to the base of the stack
4 NOT r0,#$ ;dummy value for r0
5
6 STR r0,[r13,-4]! ; stores the contents of r0 at an address -4 bytes from r13 and decrement r13 by 4
7
8 Ldf
9
10 DCD 0x433ABCDEF,0,0,0x12345678 ;stack area
11 Base DCD 0xAAAAAAA ;stack base and dummy data
12
13 END

```

[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

```

 AREA Pull_MoreEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD sp,$base point to the base of the stack
    MOV r0,$# ;dummy value for r0
    SUB r1,$r1,$# ;decrement the stack pointer to move it up
    STB r0,[r1] ;now put the item on the stack
    MOV r0,$# ;dummy value for r0
    LDR r0,[r1] ;read the item at the top of the stack
    ADD r1,$r1,$# ;increment the stack pointer
    B inf
    DCD 0x00000000,0x0,0x12345678 ;stack area
    DCD 0xFFFFFFFF,0xFFFFFFFF ;stack base and dummy data
    END

```

[25]

Pull Instruction

```

 AREA Pull_MoreEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD sp,$base point to the base of the stack
    MOV r0,$# ;dummy value for r0
    SUB r1,$r1,$# ;decrement the stack pointer to move it up
    STB r0,[r1] ;now put the item on the stack
    MOV r0,$# ;dummy value for r0
    LDR r0,[r1] ;read the item at the top of the stack
    ADD r1,$r1,$# ;increments the stack pointer
    B inf
    DCD 0x00000000,0x0,0x12345678 ;stack area
    DCD 0xFFFFFFFF,0xFFFFFFFF ;stack base and dummy data
    END

```

[26]

Pull Instruction

```

 AREA Pull_MoreEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD sp,$base point to the base of the stack
    MOV r0,$# ;dummy value for r0
    SUB r1,$r1,$# ;decrement the stack pointer to move it up
    STB r0,[r1] ;now put the item on the stack
    MOV r0,$# ;dummy value for r0
    LDR r0,[r1] ;read the item at the top of the stack
    ADD r1,$r1,$# ;increment the stack pointer
    B inf
    DCD 0x00000000,0x0,0x12345678 ;stack area
    DCD 0xFFFFFFFF,0xFFFFFFFF ;stack base and dummy data
    END

```

[27]

Pull Instruction

```

 AREA Pull_MoreEfficient, CODE, READWRITE ;make read/write because we have the stack in this area
    ADD sp,$base point to the base of the stack
    MOV r0,$# ;dummy value for r0
    SUB r1,$r1,$# ;decrement the stack pointer to move it up
    STB r0,[r1] ;now put the item on the stack
    MOV r0,$# ;dummy value for r0
    LDR r0,[r1] ;read the item at the top of the stack
    ADD r1,$r1,$# ;increments the stack pointer
    B inf
    DCD 0x00000000,0x0,0x12345678 ;stack area
    DCD 0xFFFFFFFF,0xFFFFFFFF ;stack base and dummy data
    END

```

[28]

Pull Instruction

```

    AREA Pull_Efficient, CODE, READWRITE ; make readable because we have the stack in this area
    ADD r0,base ; point to the base of the stack
    MOV r0,$4 ; dummy value for r0
    SUB r13,r13,#4 ; decrement the stack pointer to move it up
    STR r0,[r13] ; now put the item on the stack
    MOV r0,$10 ; dummy value for r0
    LDR r0,[r13],#4 ; read the item at the top of the stack
    ADD r13,r13,#4 ; increment the stack pointer
    END
  
```

[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)

```

    AREA Pull_Efficient, CODE, READWRITE ; make readable because we have the stack in this area
    ADD r0,base ; point to the base of the stack
    MOV r0,$4 ; dummy value for r0
    STR r0,[r13],#4 ; stores the contents of r0 at an address -4 bytes from r13 and decrement r13 by 4
    MOV r0,$10 ; dummy value for r0
    LDR r0,[r13],#4 ; pull and increment the stack pointer
    END
  
```

[32]

Pull Instruction (Efficient)

```

C:\Users\wahabkhan\Google Drive\PhD\fall 2013\Teaching\CS220\Tutorial\Tutorial-7\Pull_Efficient\src\main.cu
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
  Current Value
  1 AREA Pull_Efficient, CODE, READWRITE :make read/write because we have the stack in this area
  2 ADD r0,Base ;r0 point to the base of the stack
  3 MOV r0,$4 ; dummy value for r0
  4
  5 STR r0,[r13,-4] ; stores the contents of r0 at an address -4 bytes from r13 and decrement r13 by 4
  6
  7 MOV r0,$40 ; dummy value for r0
  8
  9 STR r0,[r13,-4] ; stores the contents of r0 at an address -4 bytes from r13 and decrement r13 by 4
  10
  11 LDR r0,[r13,4] ; pull and increment the stack pointer
  12 inf
  13
  14 DCD 0x00000000,0,0,0x12345678 ;stack area
  15 DCD 0xFFFFFFFF ;stack base and dummy data
  16
  17 END
  18 Base
  19
  20 R14(LR) 0x00000000
  21 R15
  22 R16
  23 R17
  24 R18
  25 R19
  26 R20
  27 R21
  28 R22
  29 R23
  30 R24
  31 R25
  32 R26
  33 R27
  34 R28
  35 R29
  36 R30
  37 R31
  38 R32
  39 R33
  40 R34
  41 R35
  42 R36
  43 R37
  44 R38
  45 R39
  46 R40
  47 R41
  48 R42
  49 R43
  50 R44
  51 R45
  52 R46
  53 R47
  54 R48
  55 R49
  56 R50
  57 R51
  58 R52
  59 R53
  60 R54
  61 R55
  62 R56
  63 R57
  64 R58
  65 R59
  66 R60
  67 R61
  68 R62
  69 R63
  70 R64
  71 R65
  72 R66
  73 R67
  74 R68
  75 R69
  76 R70
  77 R71
  78 R72
  79 R73
  80 R74
  81 R75
  82 R76
  83 R77
  84 R78
  85 R79
  86 R80
  87 R81
  88 R82
  89 R83
  90 R84
  91 R85
  92 R86
  93 R87
  94 R88
  95 R89
  96 R90
  97 R91
  98 R92
  99 R93
  100 R94
  101 R95
  102 R96
  103 R97
  104 R98
  105 R99
  106 R100
  107 R101
  108 R102
  109 R103
  110 R104
  111 R105
  112 R106
  113 R107
  114 R108
  115 R109
  116 R110
  117 R111
  118 R112
  119 R113
  120 R114
  121 R115
  122 R116
  123 R117
  124 R118
  125 R119
  126 R120
  127 R121
  128 R122
  129 R123
  130 R124
  131 R125
  132 R126
  133 R127
  134 R128
  135 R129
  136 R130
  137 R131
  138 R132
  139 R133
  140 R134
  141 R135
  142 R136
  143 R137
  144 R138
  145 R139
  146 R140
  147 R141
  148 R142
  149 R143
  150 R144
  151 R145
  152 R146
  153 R147
  154 R148
  155 R149
  156 R150
  157 R151
  158 R152
  159 R153
  160 R154
  161 R155
  162 R156
  163 R157
  164 R158
  165 R159
  166 R160
  167 R161
  168 R162
  169 R163
  170 R164
  171 R165
  172 R166
  173 R167
  174 R168
  175 R169
  176 R170
  177 R171
  178 R172
  179 R173
  180 R174
  181 R175
  182 R176
  183 R177
  184 R178
  185 R179
  186 R180
  187 R181
  188 R182
  189 R183
  190 R184
  191 R185
  192 R186
  193 R187
  194 R188
  195 R189
  196 R190
  197 R191
  198 R192
  199 R193
  200 R194
  201 R195
  202 R196
  203 R197
  204 R198
  205 R199
  206 R200
  207 R201
  208 R202
  209 R203
  210 R204
  211 R205
  212 R206
  213 R207
  214 R208
  215 R209
  216 R210
  217 R211
  218 R212
  219 R213
  220 R214
  221 R215
  222 R216
  223 R217
  224 R218
  225 R219
  226 R220
  227 R221
  228 R222
  229 R223
  230 R224
  231 R225
  232 R226
  233 R227
  234 R228
  235 R229
  236 R230
  237 R231
  238 R232
  239 R233
  240 R234
  241 R235
  242 R236
  243 R237
  244 R238
  245 R239
  246 R240
  247 R241
  248 R242
  249 R243
  250 R244
  251 R245
  252 R246
  253 R247
  254 R248
  255 R249
  256 R250
  257 R251
  258 R252
  259 R253
  260 R254
  261 R255
  262 R256
  263 R257
  264 R258
  265 R259
  266 R260
  267 R261
  268 R262
  269 R263
  270 R264
  271 R265
  272 R266
  273 R267
  274 R268
  275 R269
  276 R270
  277 R271
  278 R272
  279 R273
  280 R274
  281 R275
  282 R276
  283 R277
  284 R278
  285 R279
  286 R280
  287 R281
  288 R282
  289 R283
  290 R284
  291 R285
  292 R286
  293 R287
  294 R288
  295 R289
  296 R290
  297 R291
  298 R292
  299 R293
  300 R294
  301 R295
  302 R296
  303 R297
  304 R298
  305 R299
  306 R300
  307 R301
  308 R302
  309 R303
  310 R304
  311 R305
  312 R306
  313 R307
  314 R308
  315 R309
  316 R310
  317 R311
  318 R312
  319 R313
  320 R314
  321 R315
  322 R316
  323 R317
  324 R318
  325 R319
  326 R320
  327 R321
  328 R322
  329 R323
  330 R324
  331 R325
  332 R326
  333 R327
  334 R328
  335 R329
  336 R330
  337 R331
  338 R332
  339 R333
  340 R334
  341 R335
  342 R336
  343 R337
  344 R338
  345 R339
  346 R340
  347 R341
  348 R342
  349 R343
  350 R344
  351 R345
  352 R346
  353 R347
  354 R348
  355 R349
  356 R350
  357 R351
  358 R352
  359 R353
  360 R354
  361 R355
  362 R356
  363 R357
  364 R358
  365 R359
  366 R360
  367 R361
  368 R362
  369 R363
  370 R364
  371 R365
  372 R366
  373 R367
  374 R368
  375 R369
  376 R370
  377 R371
  378 R372
  379 R373
  380 R374
  381 R375
  382 R376
  383 R377
  384 R378
  385 R379
  386 R380
  387 R381
  388 R382
  389 R383
  390 R384
  391 R385
  392 R386
  393 R387
  394 R388
  395 R389
  396 R390
  397 R391
  398 R392
  399 R393
  400 R394
  401 R395
  402 R396
  403 R397
  404 R398
  405 R399
  406 R400
  407 R401
  408 R402
  409 R403
  410 R404
  411 R405
  412 R406
  413 R407
  414 R408
  415 R409
  416 R410
  417 R411
  418 R412
  419 R413
  420 R414
  421 R415
  422 R416
  423 R417
  424 R418
  425 R419
  426 R420
  427 R421
  428 R422
  429 R423
  430 R424
  431 R425
  432 R426
  433 R427
  434 R428
  435 R429
  436 R430
  437 R431
  438 R432
  439 R433
  440 R434
  441 R435
  442 R436
  443 R437
  444 R438
  445 R439
  446 R440
  447 R441
  448 R442
  449 R443
  450 R444
  451 R445
  452 R446
  453 R447
  454 R448
  455 R449
  456 R450
  457 R451
  458 R452
  459 R453
  460 R454
  461 R455
  462 R456
  463 R457
  464 R458
  465 R459
  466 R460
  467 R461
  468 R462
  469 R463
  470 R464
  471 R465
  472 R466
  473 R467
  474 R468
  475 R469
  476 R470
  477 R471
  478 R472
  479 R473
  480 R474
  481 R475
  482 R476
  483 R477
  484 R478
  485 R479
  486 R480
  487 R481
  488 R482
  489 R483
  490 R484
  491 R485
  492 R486
  493 R487
  494 R488
  495 R489
  496 R490
  497 R491
  498 R492
  499 R493
  500 R494
  501 R495
  502 R496
  503 R497
  504 R498
  505 R499
  506 R500
  507 R501
  508 R502
  509 R503
  510 R504
  511 R505
  512 R506
  513 R507
  514 R508
  515 R509
  516 R510
  517 R511
  518 R512
  519 R513
  520 R514
  521 R515
  522 R516
  523 R517
  524 R518
  525 R519
  526 R520
  527 R521
  528 R522
  529 R523
  530 R524
  531 R525
  532 R526
  533 R527
  534 R528
  535 R529
  536 R530
  537 R531
  538 R532
  539 R533
  540 R534
  541 R535
  542 R536
  543 R537
  544 R538
  545 R539
  546 R540
  547 R541
  548 R542
  549 R543
  550 R544
  551 R545
  552 R546
  553 R547
  554 R548
  555 R549
  556 R550
  557 R551
  558 R552
  559 R553
  560 R554
  561 R555
  562 R556
  563 R557
  564 R558
  565 R559
  566 R560
  567 R561
  568 R562
  569 R563
  570 R564
  571 R565
  572 R566
  573 R567
  574 R568
  575 R569
  576 R570
  577 R571
  578 R572
  579 R573
  580 R574
  581 R575
  582 R576
  583 R577
  584 R578
  585 R579
  586 R580
  587 R581
  588 R582
  589 R583
  590 R584
  591 R585
  592 R586
  593 R587
  594 R588
  595 R589
  596 R590
  597 R591
  598 R592
  599 R593
  600 R594
  601 R595
  602 R596
  603 R597
  604 R598
  605 R599
  606 R600
  607 R601
  608 R602
  609 R603
  610 R604
  611 R605
  612 R606
  613 R607
  614 R608
  615 R609
  616 R610
  617 R611
  618 R612
  619 R613
  620 R614
  621 R615
  622 R616
  623 R617
  624 R618
  625 R619
  626 R620
  627 R621
  628 R622
  629 R623
  630 R624
  631 R625
  632 R626
  633 R627
  634 R628
  635 R629
  636 R630
  637 R631
  638 R632
  639 R633
  640 R634
  641 R635
  642 R636
  643 R637
  644 R638
  645 R639
  646 R640
  647 R641
  648 R642
  649 R643
  650 R644
  651 R645
  652 R646
  653 R647
  654 R648
  655 R649
  656 R650
  657 R651
  658 R652
  659 R653
  660 R654
  661 R655
  662 R656
  663 R657
  664 R658
  665 R659
  666 R660
  667 R661
  668 R662
  669 R663
  670 R664
  671 R665
  672 R666
  673 R667
  674 R668
  675 R669
  676 R670
  677 R671
  678 R672
  679 R673
  680 R674
  681 R675
  682 R676
  683 R677
  684 R678
  685 R679
  686 R680
  687 R681
  688 R682
  689 R683
  690 R684
  691 R685
  692 R686
  693 R687
  694 R688
  695 R689
  696 R690
  697 R691
  698 R692
  699 R693
  700 R694
  701 R695
  702 R696
  703 R697
  704 R698
  705 R699
  706 R700
  707 R701
  708 R702
  709 R703
  710 R704
  711 R705
  712 R706
  713 R707
  714 R708
  715 R709
  716 R710
  717 R711
  718 R712
  719 R713
  720 R714
  721 R715
  722 R716
  723 R717
  724 R718
  725 R719
  726 R720
  727 R721
  728 R722
  729 R723
  730 R724
  731 R725
  732 R726
  733 R727
  734 R728
  735 R729
  736 R730
  737 R731
  738 R732
  739 R733
  740 R734
  741 R735
  742 R736
  743 R737
  744 R738
  745 R739
  746 R740
  747 R741
  748 R742
  749 R743
  750 R744
  751 R745
  752 R746
  753 R747
  754 R748
  755 R749
  756 R750
  757 R751
  758 R752
  759 R753
  760 R754
  761 R755
  762 R756
  763 R757
  764 R758
  765 R759
  766 R760
  767 R761
  768 R762
  769 R763
  770 R764
  771 R765
  772 R766
  773 R767
  774 R768
  775 R769
  776 R770
  777 R771
  778 R772
  779 R773
  780 R774
  781 R775
  782 R776
  783 R777
  784 R778
  785 R779
  786 R780
  787 R781
  788 R782
  789 R783
  790 R784
  791 R785
  792 R786
  793 R787
  794 R788
  795 R789
  796 R790
  797 R791
  798 R792
  799 R793
  800 R794
  801 R795
  802 R796
  803 R797
  804 R798
  805 R799
  806 R800
  807 R801
  808 R802
  809 R803
  810 R804
  811 R805
  812 R806
  813 R807
  814 R808
  815 R809
  816 R810
  817 R811
  818 R812
  819 R813
  820 R814
  821 R815
  822 R816
  823 R817
  824 R818
  825 R819
  826 R820
  827 R821
  828 R822
  829 R823
  830 R824
  831 R825
  832 R826
  833 R827
  834 R828
  835 R829
  836 R830
  837 R831
  838 R832
  839 R833
  840 R834
  841 R835
  842 R836
  843 R837
  844 R838
  845 R839
  846 R840
  847 R841
  848 R842
  849 R843
  850 R844
  851 R845
  852 R846
  853 R847
  854 R848
  855 R849
  856 R850
  857 R851
  858 R852
  859 R853
  860 R854
  861 R855
  862 R856
  863 R857
  864 R858
  865 R859
  866 R860
  867 R861
  868 R862
  869 R863
  870 R864
  871 R865
  872 R866
  873 R867
  874 R868
  875 R869
  876 R870
  877 R871
  878 R872
  879 R873
  880 R874
  881 R875
  882 R876
  883 R877
  884 R878
  885 R879
  886 R880
  887 R881
  888 R882
  889 R883
  890 R884
  891 R885
  892 R886
  893 R887
  894 R888
  895 R889
  896 R890
  897 R891
  898 R892
  899 R893
  900 R894
  901 R895
  902 R896
  903 R897
  904 R898
  905 R899
  906 R900
  907 R901
  908 R902
  909 R903
  910 R904
  911 R905
  912 R906
  913 R907
  914 R908
  915 R909
  916 R910
  917 R911
  918 R912
  919 R913
  920 R914
  921 R915
  922 R916
  923 R917
  924 R918
  925 R919
  926 R920
  927 R921
  928 R922
  929 R923
  930 R924
  931 R925
  932 R926
  933 R927
  934 R928
  935 R929
  936 R930
  937 R931
  938 R932
  939 R933
  940 R934
  941 R935
  942 R936
  943 R937
  944 R938
  945 R939
  946 R940
  947 R941
  948 R942
  949 R943
  950 R944
  951 R945
  952 R946
  953 R947
  954 R948
  955 R949
  956 R950
  957 R951
  958 R952
  959 R953
  960 R954
  961 R955
  962 R956
  963 R957
  964 R958
  965 R959
  966 R960
  967 R961
  968 R962
  969 R963
  970 R964
  971 R965
  972 R966
  973 R967
  974 R968
  975 R969
  976 R970
  977 R971
  978 R972
  979 R973
  980 R974
  981 R975
  982 R976
  983 R977
  984 R978
  985 R979
  986 R980
  987 R981
  988 R982
  989 R983
  990 R984
  991 R985
  992 R986
  993 R987
  994 R988
  995 R989
  996 R990
  997 R991
  998 R992
  999 R993
  1000 R994
  1001 R995
  1002 R996
  1003 R997
  1004 R998
  1005 R999
  1006 R1000
  1007 R1001
  1008 R1002
  1009 R1003
  1010 R1004
  1011 R1005
  1012 R1006
  1013 R1007
  1014 R1008
  1015 R1009
  1016 R1010
  1017 R1011
  1018 R1012
  1019 R1013
  1020 R1014
  1021 R1015
  1022 R1016
  1023 R1017
  1024 R1018
  1025 R1019
  1026 R1020
  1027 R1021
  1028 R1022
  1029 R1023
  1030 R1024
  1031 R1025
  1032 R1026
  1033 R1027
  1034 R1028
  1035 R1029
  1036 R1030
  1037 R1031
  1038 R1032
  1039 R1033
  1040 R1034
  1041 R1035
  1042 R1036
  1043 R1037
  1044 R1038
  1045 R1039
  1046 R1040
  1047 R1041
  1048 R1042
  1049 R1043
  1050 R1044
  1051 R1045
  1052 R1046
  1053 R1047
  1054 R1048
  1055 R1049
  1056 R1050
  1057 R1051
  1058 R1052
  1059 R1053
  1060 R1054
  1061 R1055
  1062 R1056
  1063 R1057
  1064 R1058
  1065 R1059
  1066 R1060
  1067 R1061
  1068 R1062
  1069 R1063
  1070 R1064
  1071 R1065
  1072 R1066
  1073 R1067
  1074 R1068
  1075 R1069
  1076 R1070
  1077 R1071
  1078 R1072
  1079 R1073
  1080 R1074
  1081 R1075
  1082 R1076
  1083 R1077
  1084 R1078
  1085 R1079
  1086 R1080
  1087 R1081
  1088 R1082
  1089 R1083
  1090 R1084
  1091 R1085
  1092 R1086
  1093 R1087
  1094 R1088
  1095 R1089
  1096 R1090
  1097 R1091
  1098 R1092
  1099 R1093
  1100 R1094
  1101 R1095
  1102 R1096
  1103 R1097
  1104 R1098
  1105 R1099
  1106 R1100
  1107 R1
```

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]

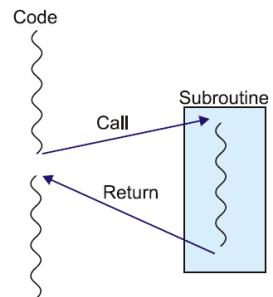
Outline

- 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



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



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

```

AREA SubroutineExample, CODE, READONLY
1    MOV R0,#4 ;set up a dummy parameter
2    MOV R1,#0
3    BL SQRL ;call SQRL
4    MOV R0,R0 ;do something with the result
5    B Loop ;stay here
6    Loop:
7    ADD R1,R1,R1 ;Increment R1 by 1
8    MOV R0,R0 ;do something with the result
9    B Loop ;stay here
10   ADD R1,R1,R1 ;Increment R1 by 1
11   MOV R0,R0 ;do something with the result
12   B Loop ;stay here
13   ADD R1,R1,R1 ;Increment R1 by 1
14   MOV R0,R0 ;do something with the result
15   B Loop ;stay here
16   ADD R1,R1,R1 ;Increment R1 by 1
17   MOV R0,R0 ;do something with the result
18   B Loop ;stay here
19   END

```

(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

```

AREA SubroutineProblemExample, CODE, READONLY
1    MOV R0,#4 ;set up a dummy parameter
2    MOV R1,#0
3    BL SQRL ;call SQRL
4    ADD R1,R1,R1 ;Increment R1 by 1
5    MOV R0,R0 ;do something with the result
6    B Loop ;stay here
7    ADD R1,R1,R1 ;Increment R1 by 1
8    MOV R0,R0 ;do something with the result
9    B Loop ;stay here
10   ADD R1,R1,R1 ;Increment R1 by 1
11   MOV R0,R0 ;do something with the result
12   B Loop ;stay here
13   ADD R1,R1,R1 ;Increment R1 by 1
14   MOV R0,R0 ;do something with the result
15   B Loop ;stay here
16   ADD R1,R1,R1 ;Increment R1 by 1
17   MOV R0,R0 ;do something with the result
18   B Loop ;stay here
19   END

```

The value should be 2. However, because we used r1 in subroutine, the value is not correct!

(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 overwriting 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 x²

ADD r0,r1,#1 ;Add 1 to get x² + 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

```

AREA SubroutineProblemsSolved, CODE, READWRITE ;make read/write because we have the stack in this area
1    ADD sp,sp,#-4          ;Save point to the base of the stack
2    MOV lr,r0              ;dummy value for link register, r14
3    MUL r1,r0,r0            ;Calculate x^2 (note - can't use source register as destination)
4    ADD r0,r1,#1             ;Add 1 to get x^2 + 1
5    LDR r1,[sp],#4           ;Restore register r1 on the stack
6    LDR lr,[sp],#4           ;Restore link register on the stack
7    MOV pc,lr                ;Return
8
9
10   Loop: B Loop ;say here
11
12   SQR1 STR lr,[sp,#-4]! ;Save link register on the stack
13   SQR1 STR r1,[sp,#-4]! ;Save register r1 on the stack
14   MUL r1,r0,r0            ;Calculate x^2 (note - can't use source register as destination)
15   ADD r0,r1,#1             ;Add 1 to get x^2 + 1
16   LDR r1,[sp],#4           ;Restore register r1 on the stack
17   LDR lr,[sp],#4           ;Restore link register on the stack
18   MOV pc,lr                ;Return
19
20   DCD 0x12345678          ;stack area
21   Base: DCD 0xAAAAAAA          ;stack base and dummy data
22
23   END

```

[58]

Subroutine – Problems

```

AREA SubroutineProblemsSolved, CODE, READWRITE ;make read/write because we have the stack in this area
1    ADD sp,sp,#-4          ;Save point to the base of the stack
2    MOV lr,r0              ;dummy value for link register, r14
3    MUL r1,r0,r0            ;Calculate x^2 (note - can't use source register as destination)
4    ADD r0,r1,#1             ;Add 1 to get x^2 + 1
5    LDR r1,[sp],#4           ;Restore register r1 on the stack
6    LDR lr,[sp],#4           ;Restore link register on the stack
7    MOV pc,lr                ;Return
8
9
10   Loop: B Loop ;say here
11
12   SQR1 STR lr,[sp,#-4]! ;Save link register on the stack
13   SQR1 STR r1,[sp,#-4]! ;Save register r1 on the stack
14   MUL r1,r0,r0            ;Calculate x^2 (note - can't use source register as destination)
15   ADD r0,r1,#1             ;Add 1 to get x^2 + 1
16   LDR r1,[sp],#4           ;Restore register r1 on the stack
17   LDR lr,[sp],#4           ;Restore link register on the stack
18   MOV pc,lr                ;Return
19
20   DCD 0x12345678          ;stack area
21   Base: DCD 0xAAAAAAA          ;stack base and dummy data
22
23   END

```

[59]

Subroutine – Problems

```

AREA SubroutineProblemsSolved, CODE, READWRITE ;make read/write because we have the stack in this area
1    ADD sp,sp,#-4          ;Save point to the base of the stack
2    MOV lr,r0              ;dummy value for link register, r14
3    MUL r1,r0,r0            ;Calculate x^2 (note - can't use source register as destination)
4    ADD r0,r1,#1             ;Add 1 to get x^2 + 1
5    LDR r1,[sp],#4           ;Restore register r1 on the stack
6    LDR lr,[sp],#4           ;Restore link register on the stack
7    MOV pc,lr                ;Return
8
9
10   Loop: B Loop ;say here
11
12   SQR1 STR lr,[sp,#-4]! ;Save link register on the stack
13   SQR1 STR r1,[sp,#-4]! ;Save register r1 on the stack
14   MUL r1,r0,r0            ;Calculate x^2 (note - can't use source register as destination)
15   ADD r0,r1,#1             ;Add 1 to get x^2 + 1
16   LDR r1,[sp],#4           ;Restore register r1 on the stack
17   LDR lr,[sp],#4           ;Restore link register on the stack
18   MOV pc,lr                ;Return
19
20   DCD 0x12345678          ;stack area
21   Base: DCD 0xAAAAAAA          ;stack base and dummy data
22
23   END

```

[60]

Subroutine – Problems

61

62

Subroutine – Problems

63

64

Subroutine – Problems

65

Subroutine – Problems

```
1 AREA SubroutineProblemsSolved, CODE, READWRITE ;make read/write because we have the stack in this area
2
3    ADM $0, #4000 ;move point to the base of the stack
4    MOV R0, #4000 ;move value for ci
5    MOV R1, #0 ;move value for r0
6    MOV R2, #1 ;set up a dummy parameter in r0
7    MOV R3, #0 ;do something with the result which is in r0
8
9    L0 Loop    B Loop ;entry here
10
11    L1:        STB 1,(R0)+#4 ;Save link register on the stack
12    L2:        STC 1,(R0)+#4 ;Save register r1 on the stack
13    L3:        ADD R0, #4 ;add 4 to stack
14    L4:        ADD R0, #4 ;add to get x + 1
15    L5:        STB 1,(R0)+#4 ;Save register r2 on the stack
16    L6:        ADD R0, #4 ;add 4 to stack
17    L7:        LDW 1,(R0)+#4 ;Get back link register on the stack
18    L8:        MOV R0,R1 ;return
19    L9:        RET
20
21    DCI $0A5ABCDEF,0,$,0;X$00000000 ;stack area
22    DCB $0A5ABCDEF,0,$,0;X$00000000 ;stack base and dummy data
23
24    END
```

66

Subroutine – Problems

67

Subroutine – Problems

The screenshot shows the assembly code for the `SubroutineProblemsSolved` function. The assembly code includes:

- Register setup: `MOV R0, #10000000`, `MOV R1, #10000000`, `MOV R2, #10000000`, `MOV R3, #10000000`, `MOV R4, #10000000`, `MOV R5, #10000000`, `MOV R6, #10000000`, `MOV R7, #10000000`, `MOV R8, #10000000`, `MOV R9, #10000000`, `MOV R10, #10000000`, `MOV R11, #10000000`, `MOV R12, #10000000`, `MOV R13, #10000000`, `MOV R14, #10000000`.
- Link register setup: `MOV R14, R13`.
- Stack pointer setup: `MOV SP, R14`.
- Parameter setup: `MOV R1, R14` (link register), `MOV R0, R14+4` (dummy parameter).
- Calculation loop: `ADD R0, R1, R0` (loop body), `Loop B Loop`.
- Result calculation: `MOV R5, R0`.
- Stack cleanup: `MOV R14, R5`.
- Return: `MOV R14, R14+4`.

The memory dump shows the stack area starting at address 0x00000000, growing downwards. Red boxes highlight the stack area and the dummy parameter. A green arrow points to the dummy parameter in the assembly code.

68

Subroutine – Problems

```

1 AREA SubroutineProblems, CODE, READONLY
2 ADD sp, #4      ;point to the base of the stack
3 MOV r1, #4      ;dummy value for r1
4 MOV r2, #4      ;dummy value for r2
5 MOV r3, #4      ;dummy value for r3
6 MOV r4, #4      ;dummy value for r4
7 MOV r5, #4      ;dummy value for r5
8 MOV r6, #4      ;dummy value for r6
9 MOV r7, #4      ;dummy value for r7
10 MOV r8, #4     ;dummy value for r8
11 MOV r9, #4     ;dummy value for r9
12 MOV R11, #4    ;Save Link register
13 LDR r1, [sp], #4    ;Save register r1 on the stack
14 LDR r2, [sp], #4    ;Save register r2 on the stack
15 LDR r3, [sp], #4    ;Save register r3 on the stack
16 LDR r4, [sp], #4    ;Save register r4 on the stack
17 LDR r5, [sp], #4    ;Save register r5 on the stack
18 LDR r6, [sp], #4    ;Save register r6 on the stack
19 LDR r7, [sp], #4    ;Save register r7 on the stack
20 LDR r8, [sp], #4    ;Save register r8 on the stack
21 LDR r9, [sp], #4    ;Save register r9 on the stack
22 LDR R11, [sp], #4  ;Restore Link register on the stack
23 MOV pc, lr       ;Return
24 END

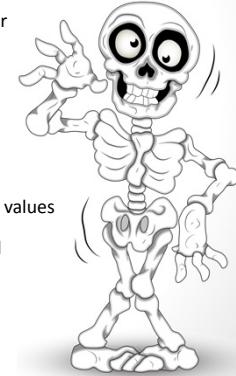
```

[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]

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2
3 ADD sp, #0x10 ;point to the base of the stack
4 MOV r0, #0xABCD ;dummy value for r0
5 MOV r1, #0x1234 ;dummy value for r1
6 MOV r2, #0x4321 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8
9 Loop
10    B Loop ;stay here
11 SQR1
12    STMFD sp!, {r0,r1,r2} ;save r0, r1, r2 on the stack
13    MOV r0, #0x4321 ;let's do something pointless
14    MOV r1, #0x1234 ;let's do something pointless
15    ADD r2, r0,r1 ;load r0 and r1 and put the result in r3
16    LDMFD sp!, {r0,r1,r2} ;pull r0, r1, r2 off the stack
17
18    DCD 0xA5A5A5A5,0,0,0x01234567 ;stack area
19    Base 0xA5A5A5A5 ;stack base and dummy data
20
21 END

```

Subroutine – Efficient push/pull registers

[74]

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2
3 ADD sp, #0x10 ;point to the base of the stack
4 MOV r0, #0xABCD ;dummy value for r0
5 MOV r1, #0x1234 ;dummy value for r1
6 MOV r2, #0x4321 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8
9 Loop
10    B Loop ;stay here
11 SQR1
12    STMFD sp!, {r0,r1,r2} ;save r0, r1, r2 on the stack
13    MOV r0, #0x4321 ;let's do something pointless
14    MOV r1, #0x1234 ;let's do something pointless
15    ADD r2, r0,r1 ;load r0 and r1 and put the result in r3
16    LDMFD sp!, {r0,r1,r2} ;pull r0, r1, r2 off the stack
17
18    DCD 0xA5A5A5A5,0,0,0x01234567 ;stack area
19    Base 0xA5A5A5A5 ;stack base and dummy data
20
21 END

```

Subroutine – Efficient push/pull registers

[75]

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2
3 ADD sp, #0x10 ;point to the base of the stack
4 MOV r0, #0xABCD ;dummy value for r0
5 MOV r1, #0x1234 ;dummy value for r1
6 MOV r2, #0x4321 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8
9 Loop
10    B Loop ;stay here
11 SQR1
12    STMFD sp!, {r0,r1,r2} ;save r0, r1, r2 on the stack
13    MOV r0, #0x4321 ;let's do something pointless
14    MOV r1, #0x1234 ;let's do something pointless
15    ADD r2, r0,r1 ;load r0 and r1 and put the result in r3
16    LDMFD sp!, {r0,r1,r2} ;pull r0, r1, r2 off the stack
17
18    DCD 0xA5A5A5A5,0,0,0x01234567 ;stack area
19    Base 0xA5A5A5A5 ;stack base and dummy data
20
21 END

```

Subroutine – Efficient push/pull registers

[76]

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2
3 ADD sp, #0x10 ;point to the base of the stack
4 MOV r0, #0xABCD ;dummy value for r0
5 MOV r1, #0x1234 ;dummy value for r1
6 MOV r2, #0x4321 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8
9 Loop
10    B Loop ;stay here
11 SQR1
12    STMFD sp!, {r0,r1,r2} ;save r0, r1, r2 on the stack
13    MOV r0, #0x4321 ;let's do something pointless
14    MOV r1, #0x1234 ;let's do something pointless
15    ADD r2, r0,r1 ;load r0 and r1 and put the result in r3
16    LDMFD sp!, {r0,r1,r2} ;pull r0, r1, r2 off the stack
17
18    DCD 0xA5A5A5A5,0,0,0x01234567 ;stack area
19    Base 0xA5A5A5A5 ;stack base and dummy data
20
21 END

```

Subroutine – Efficient push/pull registers

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2 ADD sp, #0x00000008 ;point to the base of the stack
3 MOV r0, #0x00000000 ;dummy value for r0
4 MOV r1, #0x00000000 ;dummy value for r1
5 MOV r2, #0x00000000 ;dummy value for r2
6 MOV r3, #0x00000000 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8 Loop:
9    B Loop ;stay here
10   LDRB r0, [r0, #0x00] ;move r0, cl, lr on the stack
11   STDRB r0!, [r0, #0x00] ;move r0, cl, lr on the stack
12   MOV r1, #0x422 ;let's do something pointless
13   MOV r2, #0x422 ;let's do something pointless
14   ADD r3, r1, r2 ;add r3 and r1 and put the result in r3
15   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
16   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
17   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
18   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
19   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
20   DCD 0xA5A5A5A5, 0x0, 0x0, 0x012345678 ;stack base and dummy data
21   END

```

Base Start of Stack Area [77]

Subroutine – Efficient push/pull registers

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2 ADD sp, #0x00000008 ;point to the base of the stack
3 MOV r0, #0x00000000 ;dummy value for r0
4 MOV r1, #0x00000000 ;dummy value for r1
5 MOV r2, #0x00000000 ;dummy value for r2
6 MOV r3, #0x00000000 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8 Loop:
9    B Loop ;stay here
10   LDRB r0, [r0, #0x00] ;move r0, cl, lr on the stack
11   STDRB r0!, [r0, #0x00] ;move r0, cl, lr on the stack
12   MOV r1, #0x422 ;let's do something pointless
13   MOV r2, #0x422 ;let's do something pointless
14   ADD r3, r1, r2 ;add r3 and r1 and put the result in r3
15   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
16   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
17   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
18   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
19   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
20   DCD 0xA5A5A5A5, 0x0, 0x0, 0x012345678 ;stack base and dummy data
21   END

```

Base Start of Stack Area [78]

Subroutine – Efficient push/pull registers

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2 ADD sp, #0x00000008 ;point to the base of the stack
3 MOV r0, #0x00000000 ;dummy value for r0
4 MOV r1, #0x00000000 ;dummy value for r1
5 MOV r2, #0x00000000 ;dummy value for r2
6 MOV r3, #0x00000000 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8 Loop:
9    B Loop ;stay here
10   LDRB r0, [r0, #0x00] ;move r0, cl, lr on the stack
11   STDRB r0!, [r0, #0x00] ;move r0, cl, lr on the stack
12   MOV r1, #0x422 ;let's do something pointless
13   MOV r2, #0x422 ;let's do something pointless
14   ADD r3, r1, r2 ;add r3 and r1 and put the result in r3
15   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
16   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
17   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
18   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
19   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
20   DCD 0xA5A5A5A5, 0x0, 0x0, 0x012345678 ;stack base and dummy data
21   END

```

Base Start of Stack Area [79]

Subroutine – Efficient push/pull registers

```

1 AREA BlockMove_CODE, READWRITE ;make read/write because we have the stack in this area
2 ADD sp, #0x00000008 ;point to the base of the stack
3 MOV r0, #0x00000000 ;dummy value for r0
4 MOV r1, #0x00000000 ;dummy value for r1
5 MOV r2, #0x00000000 ;dummy value for r2
6 MOV r3, #0x00000000 ;dummy value for link register, r14
7 BL SQR1 ;call Test
8 Loop:
9    B Loop ;stay here
10   LDRB r0, [r0, #0x00] ;move r0, cl, lr on the stack
11   STDRB r0!, [r0, #0x00] ;move r0, cl, lr on the stack
12   MOV r1, #0x422 ;let's do something pointless
13   MOV r2, #0x422 ;let's do something pointless
14   ADD r3, r1, r2 ;add r3 and r1 and put the result in r3
15   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
16   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
17   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
18   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
19   LDMFD r0!, {r0, r1, r2} ;push r0, cl, lr off the stack
20   DCD 0xA5A5A5A5, 0x0, 0x0, 0x012345678 ;stack base and dummy data
21   END

```

Base Start of Stack Area [80]

Subroutine – Efficient push/pull registers

81

Subroutine – Efficient push/pull registers

Start of Stack Area

Subroutine – Efficient push/pull registers

83

Subroutine - Efficient push/pull registers

Start of Stack Area

Subroutine – Efficient push/pull registers

- The skeleton of a typical **Efficient** pushing/pulling to a stack in a subroutine (call/return):

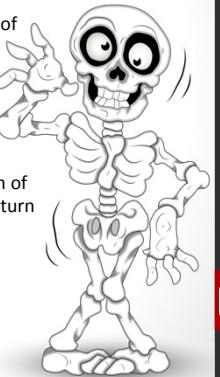
- STMFD sp!,{r0,r1,lr}

Save a bunch of registers in memory

-

- LDMFD sp!,{r0,r1,pc}

Restore a bunch of registers and return



[85]

Outline

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

[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



[89]

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



[90]

