PCB: Needs 47k Ω Resistor to pull the RESET line HIGH.

Lab1: Checkoff is this week! We don’t need to sign your handin, we will do this digitally directly on Canvas.

The handin is due on Canvas based on the end of the session you signed up for.
The Dreaded Minute Quiz
Minute Quiz

• Where is the stack in relation to other memory (such as the heap)?

• Which direction does the stack grow?

• What is at this hex location 0x20008000?
Minute Quiz

• Where is the stack in relation to other memory (such as the heap)?

• Stack is between 2 other memory locations. The heap can be placed in either “other” location
Minute Quiz

- Which direction does the stack grow?
  - Stack always grows downward from the initial location.
Minute Quiz

- What is at this hex location 0x20008000?
- This is the initial value of the Stack Pointer.
Minute Quiz

- What is at this hex location 0x20008000?
- This is the initial value of the Stack Pointer
• push \{reg\_list\}: \(\text{reg\_list}\) is stored on the stack in
  – Numerical order
  – With lowest numbered register at lowest memory address
• pop \{reg\_list\} : \(\text{reg\_list}\) is stored on the stack in
  – Numerical order
  – With lowest numbered register at lowest memory address
  – If PC is in \(\text{reg\_list}\), causes branch to addr popped off stack

• Examples
  – push \{r0, r4-r7\}
  – push \{r2, lr\}
  – pop \{r0, r1\}

• ARM stack is full-descending so a push causes the SP to hold a lower address.
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

```
mov r2, #100
movw r1, #30
movt r1, #31
strh r1, [r2], #4
str r1, [r2, 1]
sub r2, #4
strb r1, [r2], #2
strb r2, [r2, 3]
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x00</td>
</tr>
<tr>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x00</td>
</tr>
<tr>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>107</td>
<td>0x00</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

```assembly
mov r2, #100    - r2 = 100
movw r1, #30
movt r1, #31
strh r1, [r2], #4
str r1, [r2, 1]
sub r2, #4
strb r1, [r2], #2
strb r2, [r2, 3]
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x00</td>
</tr>
<tr>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x00</td>
</tr>
<tr>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>107</td>
<td>0x00</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

```
mov r2, #100       - r2 = 100
movw r1, #30       - r1 = 0x1E
movt r1, #31
strh r1, [r2], #4
str r1, [r2, 1]!
sub r2, #4
strb r1, [r2], #2
strb r2, [r2, 3]
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x00</td>
</tr>
<tr>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x00</td>
</tr>
<tr>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>107</td>
<td>0x00</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x00</td>
</tr>
<tr>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x00</td>
</tr>
<tr>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>107</td>
<td>0x00</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>

```c
mov r2, #100  - r2 = 100
movw r1, #30  - r1 = 0x1E
movt r1, #31  - r1 = 0x001F001E
strh r1, [r2], #4
str r1, [r2, 1]
sub r2, #4
strb r1, [r2], #2
strb r2, [r2, 3]
```
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov r2, #100</td>
<td>100</td>
<td>0x1E</td>
</tr>
<tr>
<td>movw r1, #30</td>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>movt r1, #31</td>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>strh r1, [r2], #4</td>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>sub r2, #4</td>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>strb r1, [r2], #2</td>
<td>105</td>
<td>0x00</td>
</tr>
<tr>
<td>strb r2, [r2, 3]</td>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>str r1, [r2, 1]!</td>
<td>107</td>
<td>0x00</td>
</tr>
<tr>
<td>strb r2, [r2, 3]</td>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

\[
\begin{align*}
\text{mov r2, #100} & \quad r2 = 100 \\
\text{movw r1, #30} & \quad r1 = 0x1E \\
\text{movt r1, #31} & \quad r1 = 0x001F001E \\
\text{strh r1, [r2], #4} & \quad [100] = 0x1E, [101] = 0x00, r2 = 104 \\
\text{str r1, [r2, 1]} & \quad r2 = 105, [105] = 0x1E, [106] = 0, [107] = 0x1F [108] = 0 \\
\text{sub r2, #4} & \\
\text{strb r1, [r2], #2} & \\
\text{strb r2, [r2, 3]} &
\end{align*}
\]

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x1E</td>
</tr>
<tr>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x1E</td>
</tr>
<tr>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>107</td>
<td>0x1F</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

```
mov r2, #100           - r2 = 100
movw r1, #30           - r1 = 0x1E
movt r1, #31           - r1 = 0x001F001E
strh r1, [r2], #4      - [100] = 0x1E, [101] = 0x00, r2 = 104
str r1, [r2, 1]!       - r2=105, [105]=0x1E,
                        [106]=0, [107]=0x1F [108]=0
sub r2, #4             - r2=101
strb r1, [r2], #2      
strb r2, [r2, 3]       
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x1E</td>
</tr>
<tr>
<td>101</td>
<td>0x00</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x1E</td>
</tr>
<tr>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>107</td>
<td>0x1F</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov r2, #100</td>
<td>100</td>
<td>0x1E</td>
</tr>
<tr>
<td>movw r1, #30</td>
<td>101</td>
<td>0x1E</td>
</tr>
<tr>
<td>movt r1, #31</td>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>strh r1, [r2], #4</td>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>str r1, [r2, 1]!</td>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>sub r2, #4</td>
<td>105</td>
<td>0x1E</td>
</tr>
<tr>
<td>strb r1, [r2], #2</td>
<td>106</td>
<td>0x00</td>
</tr>
<tr>
<td>strb r2, [r2, 3]</td>
<td>107</td>
<td>0x1F</td>
</tr>
</tbody>
</table>

- \( r2 = 100 \)
- \( r1 = 0x1E \)
- \( r1 = 0x001F001E \)
- \([100] = 0x1E, [101] = 0x00\)
- \( r2 = 104 \)
- \( r2 = 105, [105] = 0x1E, [106] = 0, [107] = 0x1F [108] = 0 \)
- \( r2 = 101 \)
- \( [101] = 0x1E, r2 = 103 \)
Write down the memory content at the end of the execution of the following program. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset.

```
mov r2, #100  - r2 = 100
movw r1, #30   - r1 = 0x1E
movt r1, #31   - r1 = 0x001F001E
strh r1, [r2], #4 - [100] = 0x1E, [101] = 0x00, r2 = 104
str r1, [r2, 1]! - r2=105, [105]=0x1E, [106]=0, [107]=0x1F [108]=0
sub r2, #4    - r2=101
strb r1, [r2], #2 - [101]=0x1E, r2=103
strb r2, [r2, 3] - [106]=0x67
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0x1E</td>
</tr>
<tr>
<td>101</td>
<td>0x1E</td>
</tr>
<tr>
<td>102</td>
<td>0x00</td>
</tr>
<tr>
<td>103</td>
<td>0x00</td>
</tr>
<tr>
<td>104</td>
<td>0x00</td>
</tr>
<tr>
<td>105</td>
<td>0x1E</td>
</tr>
<tr>
<td>106</td>
<td>0x67</td>
</tr>
<tr>
<td>107</td>
<td>0x1F</td>
</tr>
<tr>
<td>108</td>
<td>0x00</td>
</tr>
</tbody>
</table>
MOV R0, #9
loop1 CBZ R0, loop1exit
BL myfunc
SUB R0, #1
B loop1
loop1exit
MOV R0, #9 ; Set counter i = 9
loop1 CBZ R0, loop1exit
BL myfunc
SUB R0, #1
B loop1
loop1exit
MOV R0, #9 ; Set counter i = 9
loop1 CBZ R0, loop1exit ; if loop counter = 0 then exit the loop
BL myfunc
SUB R0, #1
B loop1
loop1exit
MOV R0, #9 ; Set counter i = 9
loop1  CBZ R0, loop1exit ; if loop counter = 0 then exit the loop
BL myfunc ; call a function
SUB R0, #1
B loop1
loop1exit
MOV R0, #9 ; Set counter i = 9
loop1 CBZ R0, loop1exit ; if loop counter = 0 then exit the loop
BL myfunc ; call a function
SUB R0, #1 ; loop counter decrement
B loop1
loop1exit
MOV R0, #9
loop1 CBZ R0, loop1exit
BL myfunc
SUB R0, #1
B loop1
loop1exit

; Set counter i = 9
; if loop counter = 0 then exit the loop
; call a function
; loop counter decrement
; next loop
Assembler Language

MOV R0, #9 ; Set counter i = 9
loop1 CBZ R0, loop1exit ; if loop counter = 0 then exit the loop
BL myfunc ; call a function
SUB R0, #1 ; loop counter decrement
B loop1 ; next loop
loop1exit

i = 9;
while (i != 0 )
{
    myfunc();
    i--;
}


C Code -> Assembler Language

main.c:

#include "stdint.h"

int32_t d(int32_t a, int32_t b)
{
    if(b == 0)
        return 0;
    return a/b;
}

int main() {
    volatile uint8_t h = 1, i = 0;
    return d(h, i);
}
Assembler Language

00008000 <d>:
8000: b111 cbz r1, 8008 <d+0x8>
8002: fb90 f0f1 sdiv r0, r0, r1
8006: 4770 bx lr
8008: f04f 0000 mov.w r0, #0
800c: 4770 bx lr
800e: bf00 nop

00008010 <main>:
8010: b500 push {lr}
8012: b083 sub sp, #12
8014: f04f 0301 mov.w r3, #1
8018: f88d 3007 strb.w r3, [sp, #7]
801c: f04f 0000 mov.w r0, #0
8020: f88d 0006 strb.w r0, [sp, #6]
8024: f89d 0007 ldrb.w r0, [sp, #7]
8028: f89d 1006 ldrb.w r1, [sp, #6]
802c: f7ff ffe8 bl 8000 <d>
8030: b003 add sp, #12
8032: bd00 pop {pc}
00008000 <main>:
  8000:  2000    movs    r0, #0
  8002:  4770    bx     lr

00008004 <d>:
  8004:  b111    cbz     r1, 800c <d+0x8>
  8006:  fb90 f0f1 sdiv    r0, r0, r1
  800a:  4770    bx     lr
  800c:  4608    mov     r0, r1
  800e:  4770    bx     lr
Assembler Language

00008000 <d>:
8000:   b111       cbz    r1, 8008 <d+0x8>
8002:   fb90 f0f1 sdiv r0, r0, r1
8006:   4770       bx     lr
8008:   f04f 0000 mov.w r0, #0
800c:   4770       bx     lr
800e:   bf00       nop

00008010 <main>:
8010:   b500       push   {lr}
8012:   b083       sub    sp, #12
8014:   f04f 0301 mov.w r3, #1
8018:   f88d 3007 strb.w r3, [sp, #7]
801c:   f04f 0000 mov.w r0, #0
8020:   f88d 0006 strb.w r0, [sp, #6]
8024:   f89d 0007 ldrb.w r0, [sp, #7]
8028:   f89d 1006 ldrb.w r1, [sp, #6]
802c:   f7ff ffe8 bl   8000 <d>
8030:   b003       add    sp, #12
8032:   bd00       pop    {pc}
00008000 <d>:
8000:  b111       cbz  r1, 8008 <d+0x8>  
8002:  fb90 f0f1  sdiv  r0, r0, r1
8006:  4770       bx  lr
8008:  f04f 0000  mov.w  r0, #0
800c:  4770       bx  lr
800e:  bf00       nop

Compare Branch on Zero, Branch to 8008 when r1=0

00008010 <main>:
8010:  b500       push  {lr}
8012:  b083       sub  sp, #12
8014:  f04f 0301  mov.w  r3, #1
8018:  f88d 3007  strb.w  r3, [sp, #7]
801c:  f04f 0000  mov.w  r0, #0
8020:  f88d 0006  strb.w  r0, [sp, #6]
8024:  f89d 0007  ldrb.w  r0, [sp, #7]
8028:  f89d 1006  ldrb.w  r1, [sp, #6]
802c:  f7ff ffe8  bl  8000 <d>
8030:  b003       add  sp, #12
8032:  bd00       pop  {pc}
Assembler Language

00008000 <d>:

8000: b111 cbz r1, 8008 <d+0x8> Compare Branch on Zero, Branch to 8008 when r1=0
     fb90 f0f1 sdiv r0, r0, r1 Signed Divide r0=r0/r1
     4770 bx lr
8008: f04f 0000 mov.w r0, #0
800c: 4770 bx lr
800e: bf00 nop

00008010 <main>:

8010: b500 push {lr}
8012: b083 sub sp, #12
8014: f04f 0301 mov.w r3, #1
8018: f88d 3007 strb.w r3, [sp, #7]
801c: f04f 0000 mov.w r0, #0
8020: f88d 0006 strb.w r0, [sp, #6]
8024: f89d 0007 ldrb.w r0, [sp, #7]
8028: f89d 1006 ldrb.w r1, [sp, #6]
802c: f7ff ffe8 bl 8000 <d>
8030: b003 add sp, #12
8032: bd00 pop {pc}
Assembler Language

00008000 <d>:

8000:  b111       cbz  r1, 8008 <d+0x8>  Compare Branch on Zero, Branch to 8008 when r1=0
8002:  fb90 f0f1  sdiv r0, r0, r1  Signed Divide r0=r0/r1
8006:  4770       bx lr  Branch to Link Register (Return), or MOV PC,LR
8008:  f04f 0000  mov.w r0, #0
800c:  4770       bx lr
800e:  bf00       nop

00008010 <main>:

8010:  b500       push {lr}
8012:  b083       sub sp, #12
8014:  f04f 0301  mov.w r3, #1
8018:  f88d 3007  strb.w r3, [sp, #7]
801c:  f04f 0000  mov.w r0, #0
8020:  f88d 0006  strb.w r0, [sp, #6]
8024:  f89d 0007  ldrb.w r0, [sp, #7]
8028:  f89d 1006  ldrb.w r1, [sp, #6]
802c:  f7ff ffe8  bl 8000 <d>
8030:  b003       add sp, #12
8032:  bd00       pop {pc}
Assembler Language

00008000 <d>:
8000:   b111    cbz    r1, 8008 <d+0x8>  
     Compare Branch on Zero, Branch to 8008 when r1=0
8002:   fb90 f0f1  sdiv    r0, r0, r1
     Signed Divide r0=r0/r1
8006:    4770    bx    lr
     Branch to Link Register (Return), or MOV PC,LR
8008:   f04f 0000  mov.w    r0, #0
800c:    4770    bx    lr
800e:    bf00    nop
     Return 0;

00008010 <main>:
8010:    b500    push    {lr}
8012:    b083    sub    sp, #12
8014:    f04f 0301  mov.w    r3, #1
8018:    f88d 3007  strb.w    r3, [sp, #7]
801c:    f04f 0000  mov.w    r0, #0
8020:    f88d 0006  strb.w    r0, [sp, #6]
8024:    f89d 0007  ldrb.w    r0, [sp, #7]
8028:    f89d 1006  ldrb.w    r1, [sp, #6]
802c:    f7ff ffe8  bl    8000 <d>
     Compare Branch on Zero, Branch to 8008 when r1=0
8030:    b003    add    sp, #12
8032:    bd00    pop    {pc}
Assembler Language

00008000 <d>:

8000:   b111 cbz r1, 8008 <d+0x8>  Compare Branch on Zero, Branch to 8008 when r1=0
8002:   fb90 f0f1 sdiv r0, r0, r1  Signed Divide r0=r0/r1
8006:   4770 bx lr  Branch to Link Register (Return), or MOV PC,LR
8008:   f04f 0000 mov.w r0, #0  Return 0;
800c:   4770 bx lr
800e:   bf00 nop

00008010 <main>:

8010:   b500 push {lr}  Pushing Link Register (R14)
8012:   b083 sub sp, #12
8014:   f04f 0301 mov.w r3, #1
8018:   f88d 3007 strb.w r3, [sp, #7]
801c:   f04f 0000 mov.w r0, #0
8020:   f88d 0006 strb.w r0, [sp, #6]
8024:   f89d 0007 ldrb.w r0, [sp, #7]
8028:   f89d 1006 ldrb.w r1, [sp, #6]
802c:   f7ff ffe8 bl 8000 <d>
8030:   b003 add sp, #12
8032:   bd00 pop {pc}
00008000 <d>:
8000:   b111   cbz r1, 8008 <d+0x8>  Compare Branch on Zero, Branch to 8008 when r1=0
8002:   fb90 f0f1 sdiv r0, r0, r1       Signed Divide r0=r0/r1
8006:   4770   bx lr       Branch to Link Register (Return), or MOV PC,LR
8008:   f04f 0000  mov.w r0, #0
800c:   4770   bx lr
800e:   bf00   nop

00008010 <main>:
8010:   b500   push {lr}       Pushing Link Register (R14)
8012:   b083   sub sp, #12  Subtract 12 from the Stack Pointer
8014:   f04f 0301  mov.w r3, #1
8018:   f88d 3007  strb.w r3, [sp, #7]
801c:   f04f 0000  mov.w r0, #0
8020:   f88d 0006  strb.w r0, [sp, #6]
8024:   f89d 0007  ldrb.w r0, [sp, #7]
8028:   f89d 1006  ldrb.w r1, [sp, #6]
802c:   f7ff ffe8   bl 8000 <d>  
8030:   b003   add sp, #12
8032:   bd00   pop {pc}
Assembler Language

00008000 <d>:

8000:    b111    cbz   r1, 8008 <d+0x8>
8002:    fb90 f0f1  sdiv r0, r0, r1
8006:    4770    bx    lr
8008:    f04f 0000  mov.w r0, #0
800c:    4770    bx    lr
800e:    bf00    nop

Compare Branch on Zero, Branch to 8008 when r1=0
Signed Divide r0=r0/r1
Branch to Link Register (Return), or MOV PC,LR
Return 0;

00008010 <main>:

8010:    b500    push   {lr}
8012:    b083    sub    sp, #12
8014:    f04f 0301  mov.w r3, #1
8018:    f88d 3007  strb.w r3, [sp, #7]
801c:    f04f 0000  mov.w r0, #0
8020:    f88d 0006  strb.w r0, [sp, #6]
8024:    f89d 0007  ldrb.w r0, [sp, #7]
8028:    f89d 1006  ldrb.w r1, [sp, #6]
802c:    f7ff ffe8  bl  8000 <d>
8030:    b003    add    sp, #12
8032:    bd00    pop    {pc}

Pushing Link Register (R14)
Subtract 12 from the Stack Pointer
h=1;
## Assembler Language

### 00008000 <d>:

- **8000**: `b111 cbz r1, 8008 <d+0x8>` → **Compare Branch on Zero, Branch to 8008 when r1=0**
- **8002**: `fb90 f0f1 sdiv r0, r0, r1` → **Signed Divide r0=r0/r1**
- **8006**: `4770 bx lr` → **Branch to Link Register (Return), or MOV PC,LR**
- **8008**: `f04f 0000 mov.w r0, #0` → **Return 0;**
- **800c**: `4770 bx lr`
- **800e**: `bf00 nop`

### 00008010 <main>:

- **8010**: `b500 push {lr}` → **Pushing Link Register (R14)**
- **8012**: `b083 sub sp, #12` → **Subtract 12 from the Stack Pointer**
- **8014**: `f04f 0301 mov.w r3, #1` → **h=1;**
- **8018**: `f88d 3007 strb.w r3, [sp, #7]` → **put r3 on the stack**
- **801c**: `f04f 0000 mov.w r0, #0`
- **8020**: `f88d 0006 strb.w r0, [sp, #6]`
- **8024**: `f89d 0007 ldrb.w r0, [sp, #7]`
- **8028**: `f89d 1006 ldrb.w r1, [sp, #6]`
- **802c**: `f7ff ffe8 bl 8000 <d>`
- **8030**: `b003 add sp, #12`
- **8032**: `bd00 pop {pc}`
00008000 <d>:
8000:   b111  cbz  r1, 8008 <d+0x8>  Compare Branch on Zero, Branch to 8008 when r1=0
8002:   fb90 f0f1  sdiv  r0, r0, r1  Signed Divide r0=r0/r1
8006:   4770  bx  lr  Branch to Link Register (Return), or MOV PC,LR
8008:   f0f0 0000  mov.w  r0, #0
800c:   4770  bx  lr
800e:   bf00  nop

00008010 <main>:
8010:   b500  push  {lr}  Pushing Link Register (R14)
8012:   b083  sub  sp, #12  Subtract 12 from the Stack Pointer
8014:   f0f0 0301  mov.w  r3, #1  h=1;
8018:   f88d 3007  strb.w  r3, [sp, #7]  put r3 on the stack
801c:   f0f0 0000  mov.w  r0, #0  i=0;
8020:   f88d 0006  strb.w  r0, [sp, #6]
8024:   f89d 0007  ldrb.w  r0, [sp, #7]
8028:   f89d 1006  ldrb.w  r1, [sp, #6]
802c:   f7ff ffe8  bl  8000 <d>
8030:   b003  add  sp, #12
8032:   bd00  pop  {pc}
### Assembler Language

#### 00008000 <d>:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>b111  cbz r1, 8008 &lt;d+0x8&gt;</td>
<td>Compare Branch on Zero, Branch to 8008 when r1=0</td>
</tr>
<tr>
<td>8002</td>
<td>fb90 f0f1 sdiv r0, r0, r1</td>
<td>Signed Divide r0=r0/r1</td>
</tr>
<tr>
<td>8006</td>
<td>4770  bx lr</td>
<td>Branch to Link Register (Return), or MOV PC,LR</td>
</tr>
<tr>
<td>8008</td>
<td>f04f 0000 mov.w r0, #0</td>
<td>Return 0;</td>
</tr>
<tr>
<td>800c</td>
<td>4770  bx lr</td>
<td></td>
</tr>
<tr>
<td>800e</td>
<td>bf00  nop</td>
<td></td>
</tr>
</tbody>
</table>

#### 00008010 <main>:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8010</td>
<td>b500  push {lr}</td>
<td>Pushing Link Register (R14)</td>
</tr>
<tr>
<td>8012</td>
<td>b083  sub sp, #12</td>
<td>Subtract 12 from the Stack Pointer</td>
</tr>
<tr>
<td>8014</td>
<td>f04f 0301 mov.w r3, #1</td>
<td>h=1;</td>
</tr>
<tr>
<td>8018</td>
<td>f88d 3007 strb.w r3, [sp, #7]</td>
<td>put r3 on the stack</td>
</tr>
<tr>
<td>801c</td>
<td>f04f 0000 mov.w r0, #0</td>
<td>i=0;</td>
</tr>
<tr>
<td>8020</td>
<td>f88d 0006 strb.w r0, [sp, #6]</td>
<td>put r0 on the stack</td>
</tr>
<tr>
<td>8024</td>
<td>f89d 0007 ldrb.w r0, [sp, #7]</td>
<td></td>
</tr>
<tr>
<td>8028</td>
<td>f89d 1006 ldrb.w r1, [sp, #6]</td>
<td></td>
</tr>
<tr>
<td>802c</td>
<td>f7ff ffe8 bl 8000 &lt;d&gt;</td>
<td></td>
</tr>
<tr>
<td>8030</td>
<td>b003  add sp, #12</td>
<td></td>
</tr>
<tr>
<td>8032</td>
<td>bd00  pop {pc}</td>
<td></td>
</tr>
</tbody>
</table>
Assembler Language

00008000 <d>:

8000:    b111       cbz  r1, 8008 <d+0x8>  
8002:    fb90 f0f1  sdiv r0, r0, r1  
8006:    4770       bx  lr  
8008:    f04f 0000  mov.w r0, #0  
800c:    4770       bx  lr  
800e:    bf00       nop  

Compare Branch on Zero, Branch to 8008 when r1=0
Signed Divide r0=r0/r1
Branch to Link Register (Return), or MOV PC,LR

Return 0;

00008010 <main>:

8010:    b500       push  {lr}  
8012:    b083       sub  sp, #12  
8014:    f04f 0301  mov.w r3, #1  
8018:    f88d 3007  strb.w r3, [sp, #7]  
801c:    f04f 0000  mov.w r0, #0  
8020:    f88d 0006  strb.w r0, [sp, #6]  
8024:    f89d 0007  ldrb.w r0, [sp, #7]  
8028:    f89d 1006  ldrb.w r1, [sp, #6]  
802c:    f7ff ffe8  bl  8000 <d>  
8030:    b003       add  sp, #12  
8032:    bd00       pop  {pc}  

Pushing Link Register (R14)
Subtract 12 from the Stack Pointer
h=1;
put r3 on the stack
i=0;
put r0 on the stack
get r3 from the stack
Assembler Language

00008000 <d>:
8000:   b111   cbz   r1, 8008 <d+0x8>  \textit{Compare Branch on Zero, Branch to 8008 when r1=0}
8002:   fb90 f0f1 sdiv r0, r0, r1  \textit{Signed Divide r0=r0/r1}
8006:   4770   bx   lr  \textit{Branch to Link Register (Return), or MOV PC,LR}
8008:   f04f 0000 mov.w r0, #0  \textit{Return 0;}
800c:   4770   bx   lr
800e:   bf00   nop

00008010 <main>:
8010:   b500   push  \{lr\}  \textit{Pushing Link Register (R14)}
8012:   b083   sub   sp, #12  \textit{Subtract 12 from the Stack Pointer}
8014:   f04f 0301 mov.w r3, #1  \textit{h=1;}
8018:   f88d 3007 strb.w r3, [sp, #7]  \textit{put r3 on the stack}
801c:   f04f 0000 mov.w r0, #0  \textit{i=0;}
8020:   f88d 0006 strb.w r0, [sp, #6]  \textit{put r0 on the stack}
8024:   f89d 0007 ldrb.w r0, [sp, #7]  \textit{get r3 from the stack}
8028:   f89d 1006 ldrb.w r1, [sp, #6]  \textit{get r0 from the stack}
802c:   f7ff ffe8 bl 8000 <d>
8030:   b003   add   sp, #12
8032:   bd00   pop  \{pc\}
Assembler Language

00008000 <d>:
8000:   b111  cbz  r1, 8008 <d+0x8>  \textit{Compare Branch on Zero, Branch to 8008 when } r1=0\textit{ \ Bold}\textit{ \}\textit{Signed Divide }r0=r0/r1
8002:   fb90 f0f1  sdiv  r0, r0, r1
8006:   4770  bx  lr  \textit{Branch to Link Register (Return), or MOV PC,LR}
8008:   f04f 0000  mov.w  r0, #0
800c:   4770  bx  lr
800e:   bf00  nop

00008010 <main>:
8010:   b500  push  \{lr\}  \textit{Pushing Link Register (R14)}
8012:   b083  sub  sp, #12  \textit{Subtract 12 from the Stack Pointer}
8018:   f88d 3007  strb.w  r3, [sp, #7]  \textit{put } r3\textit{ on the stack}
801c:   f04f 0000  mov.w  r0, #0  \textit{i}=0; \textit{put } r0\textit{ on the stack}
8020:   f88d 0006  strb.w  r0, [sp, #6]  \textit{get } r3\textit{ from the stack}
8024:   f89d 0007  ldrb.w  r0, [sp, #7]  \textit{get } r0\textit{ from the stack}
8028:   f89d 1006  ldrb.w  r1, [sp, #6]  \textit{d}(h,i)
802c:   f7ff ffe8  bl  8000 <d>  \textit{d}(h,i)
8030:   b003  add  sp, #12
8032:   bd00  pop  \{pc\}
00008000 <d>:
8000:  b111  cbz r1, 8008 <d+0x8>  Compare Branch on Zero, Branch to 8008 when r1=0
8002:  fb90 f0f1 sdiv r0, r0, r1  Signed Divide r0=r0/r1
8006:  4770  bx lr  Branch to Link Register (Return), or MOV PC,LR
8008:  f04f 0000  mov.w r0, #0  Return 0;
800c:  4770  bx lr
800e:  bf00  nop

00008010 <main>:
8010:  b500  push {lr}  Pushing Link Register (R14)
8012:  b083  sub sp, #12  Subtract 12 from the Stack Pointer
8014:  f04f 0301  mov.w r3, #1  h=1;
8018:  f88d 3007  strb.w r3, [sp, #7]  put r3 on the stack
801c:  f04f 0000  mov.w r0, #0  i=0;
8020:  f88d 0006  strb.w r0, [sp, #6]  put r0 on the stack
8024:  f89d 0007  ldrb.w r0, [sp, #7]  get r3 from the stack
8028:  f89d 1006  ldrb.w r1, [sp, #6]  get r0 from the stack
802c:  f7ff ffe8  bl 8000 <d>  d(h,i)
8030:  b003  add sp, #12  Add 12 from the Stack Pointer (R13)
8032:  bd00  pop {pc}
00008000 <d>:

8000:   b111  cbz    r1, 8008 <d+0x8>  Compare Branch on Zero, Branch to 8008 when r1=0
8002:   fb90  f0f1  sdiv  r0, r0, r1  Signed Divide r0=r0/r1
8006:   4770  bx     lr  Branch to Link Register (Return), or MOV PC,LR
8008:   f04f  0000  mov.w  r0, #0  Return 0;
800c:   4770  bx     lr
800e:   bf00  nop

00008010 <main>:

8010:   b500  push   {lr}  Pushing Link Register (R14)
8012:   b083  sub    sp, #12  Subtract 12 from the Stack Pointer
8014:   f04f  0301  mov.w  r3, #1  h=1;
8018:   f88d  3007  strb.w r3, [sp, #7]  put r3 on the stack
801c:   f04f  0000  mov.w  r0, #0  i=0;
8020:   f88d  0006  strb.w r0, [sp, #6]  put r0 on the stack
8024:   f89d  0007  ldrb.w r0, [sp, #7]  get r3 from the stack
8028:   f89d  1006  ldrb.w r1, [sp, #6]  get r0 from the stack
802c:   f7ff ffe8  bl    8000 <d>  d(h,i)
8030:   b003  add    sp, #12  Add 12 from the Stack Pointer (R13)
8032:   bd00  pop    {pc}  Pop Program Counter (R15)
if (R1 < R2) then
if (R1 < R2) then
    R2 = R2 - R1
if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
if (R1 < R2) then
   R2 = R2 - R1
   R2 = R2 / 2
else
   R1 = R1 - R2
   R1 = R1 / 2
if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
  R1 = R1 - R2
  R1 = R1 / 2
if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
    R1 = R1 / 2
A read transaction

if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
    R1 = R1 / 2
if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
  R1 = R1 - R2
  R1 = R1 / 2
A read transaction

if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
    R1 = R1 / 2
A read transaction

if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
    R1 = R1 / 2
A read transaction

if (R1 < R2) then
   R2 = R2 - R1
   R2 = R2 / 2
else
   R1 = R1 - R2
   R1 = R1 / 2
CMP R1, R2 ; If R1 < R2 (less then)

if (R1 < R2) then

R2 = R2 - R1
R2 = R2 / 2

else

R1 = R1 - R2
R1 = R1 / 2
A read transaction

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2

if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
  R1 = R1 - R2
  R1 = R1 / 2
A read transaction

if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
    R1 = R1 / 2

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2
; (indicated by T)
A read transaction

if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
  R1 = R1 - R2
  R1 = R1 / 2

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2
            ; (indicated by T)
            ; else execute instruction 3 and 4
A read transaction

if (R1 < R2) then
    R2 = R2 - R1
    R2 = R2 / 2
else
    R1 = R1 - R2
    R1 = R1 / 2

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2
; (indicated by T)
; else execute instruction 3 and 4
; (indicated by E)
A read transaction

if (R1 < R2) then
R2 = R2 - R1
R2 = R2 / 2
else
R1 = R1 - R2
R1 = R1 / 2

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2
 ; (indicated by T)
 ; else execute instruction 3 and 4
 ; (indicated by E)
SUBLT.W R2,R1 ; 1st instruction
A read transaction

if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
  R1 = R1 - R2
  R1 = R1 / 2

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2
          ; (indicated by T)
          ; else execute instruction 3 and 4
          ; (indicated by E)
SUBLT.W R2,R1 ; 1st instruction
LSRLT.W R2,#1 ; 2nd instruction
A read transaction

if \( R1 < R2 \) then
\[
\begin{align*}
R2 &= R2 - R1 \\
R2 &= R2 / 2
\end{align*}
\]
else
\[
\begin{align*}
R1 &= R1 - R2 \\
R1 &= R1 / 2
\end{align*}
\]

CMP R1, R2  ; If \( R1 < R2 \) (less then)
ITTEE LT  ; then execute instruction 1 and 2
; (indicated by T)
; else execute instruction 3 and 4
; (indicated by E)

SUBLT.W R2,R1  ; 1st instruction
LSRLT.W R2,#1  ; 2nd instruction
SUBGE.W R1,R2  ; 3rd instruction (notice the GE is
A read transaction

if \( R1 < R2 \) then
  \[ R2 = R2 - R1 \]
  \[ R2 = R2 / 2 \]
else
  \[ R1 = R1 - R2 \]
  \[ R1 = R1 / 2 \]

CMP R1, R2 ; If \( R1 < R2 \) (less then)
ITTEE LT ; then execute instruction 1 and 2
; (indicated by T)
; else execute instruction 3 and 4
; (indicated by E)
SUBLT.W R2,R1 ; 1st instruction
LSRLT.W R2,#1 ; 2nd instruction
SUBGE.W R1,R2 ; 3rd instruction (notice the GE is
; opposite of LT)
A read transaction

if (R1 < R2) then
  R2 = R2 - R1
  R2 = R2 / 2
else
  R1 = R1 - R2
  R1 = R1 / 2

CMP R1, R2 ; If R1 < R2 (less then)
ITTEE LT ; then execute instruction 1 and 2
           ; (indicated by T)
           ; else execute instruction 3 and 4
           ; (indicated by E)
SUBLT.W R2,R1 ; 1st instruction
LSRLT.W R2,#1 ; 2nd instruction
SUBGE.W R1,R2 ; 3rd instruction (notice the GE is
               ; opposite of LT)
LSRGE.W R1,#1 ; 4th instruction
A simple ABI routine

- int bob(int a, int b)
  - returns $a^2 + b^2$

- Instructions you might need
  - add adds two values
  - mul multiplies two values
  - mla multiply and accumulate
  - bx branch to register
int bob(int a, int b)
  - returns $a^2 + b^2$
Some disassembly

• 0x20000490 <bob>:   push {r7}
• 0x20000492 <bob+2>:  sub sp, #20
• 0x20000494 <bob+4>:  add r7, sp, #0
• 0x20000496 <bob+6>:  str r0, [r7, #4]
• 0x20000498 <bob+8>:  str r1, [r7, #0]
• x=a*a;
• 0x2000049a <bob+10>: ldr r3, [r7, #4]
• 0x2000049c <bob+12>: ldr r2, [r7, #4]
• 0x2000049e <bob+14>: mul.w r3, r2, r3
• 0x200004a2 <bob+18>: str r3, [r7, #8]
• y=b*b;
• 0x200004a4 <bob+20>: ldr r3, [r7, #0]
• 0x200004a6 <bob+22>: ldr r2, [r7, #0]
• 0x200004a8 <bob+24>: mul.w r3, r2, r3
• 0x200004ac <bob+28>: str r3, [r7, #12]
• x=x+y;
• 0x200004ae <bob+30>: ldr r2, [r7, #8]
• 0x200004b0 <bob+32>: ldr r3, [r7, #12]
• 0x200004b2 <bob+34>: add r3, r2
• 0x200004b4 <bob+36>: str r3, [r7, #8]
• return(x);
• 0x200004b6 <bob+38>: ldr r3, [r7, #8]
• }
• 0x200004b8 <bob+40>: mov r0, r3
• 0x200004ba <bob+42>: add.w r7, r7, #20
• 0x200004be <bob+46>: mov sp, r7
• 0x200004c0 <bob+48>: pop {r7}
• 0x200004c2 <bob+50>: bx lr

int bob(int a, int b)
{
    int x, y;
    x=a*a;
    y=b*b;
    x=x+y;
    return(x);
}
A read transaction
A read transaction

- Say initiator wants to read location 0x24
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
  – Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
  – Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  – Target sees read request.
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
  – Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  – Target sees read request.
  – Target drives data onto data bus.
A read transaction

- Say initiator wants to read location 0x24
  - Initiator sets ADS=0x24, CMD=0.
  - Initiator then sets REQ# to low. (why do we need a delay? How much of a delay?)
  - Target sees read request.
  - Target drives data onto data bus.
  - Target then sets ACK# to low.
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
  – Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  – Target sees read request.
  – Target drives data onto data bus.
  – Target *then* sets ACK# to low.
  – Initiator grabs the data from the data bus.
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
  – Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  – Target sees read request.
  – Target drives data onto data bus.
  – Target *then* sets ACK# to low.
  – Initiator grabs the data from the data bus.
  – Initiator sets REQ# to high, stops driving ADS and CMD
A read transaction

• Say initiator wants to read location 0x24
  – Initiator sets ADS=0x24, CMD=0.
  – Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  – Target sees read request.
  – Target drives data onto data bus.
  – Target *then* sets ACK# to low.
  – Initiator grabs the data from the data bus.
  – Initiator sets REQ# to high, stops driving ADS and CMD
  – Target stops driving data, sets ACK# to high terminating the transaction
ADS[7:0] 0x24

CMD

Data[7:0] 0x55

REQ#

ACK#

A B C D E F G HI
A write transaction
(write 0xF4 to location 0x31)
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator \textit{then} sets REQ\# to low.
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator then sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
- Target then sets ACK# to low.
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
- Target *then* sets ACK# to low.
- Initiator sets REQ# to high & stops driving other lines.
A write transaction
(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
- Target *then* sets ACK# to low.
- Initiator sets REQ# to high & stops driving other lines.
- Target sets ACK# to high terminating the transaction
Projects
– Start thinking about what you want your project to be.
- Start thinking about what you want your project to be.
– Start thinking about what you want your project to be.

– Talk with others about ideas and constraints.
– Start thinking about what you want your project to be.

– Talk with others about ideas and constraints.
- Start thinking about what you want your project to be.

- Talk with others about ideas and constraints.

- Must be Safe!!!!!!