Tutorial 3 Sample Answer

Q1 Variables assigned to registers in line with RISC-1 register usage before code generation.

```
// int g = 4                  ; global variable g:r9

        add r0, #4, r9        ; need to initialise r9

// int min(int a, int b, int c) {   ; parameters a:r26 b:r27 c:r28
//   int v = a;
//   if (b < v)
//        v = b;
//   if (c < v)
//        v = c;
//    return v;               ; use r1 for result,  r25 for return address
// }

// optimised

min:    add     r26, r0, r1     ; use r1 for local v (function result returned in r1)
        sub     r27, r1, r0 {C} ; b < v
        jge     min0            ;
        xor     r0, r0, r0      ; nop in delay slot
        add     r27, r0,  r1    ; v = b
min0:   sub     r28, r1, r0 {C} ; c < v
        jge     min1            ;
        xor     r0, r0, r0      ; nop in delay slot
        add     r28, r0,  r1    ; v = b
min1:   ret     r25, 0          ; return
        xor     r0, r0, r0      ; nop in delay slot
```

NOT possible to remove any of the nops in the delay slots

```
//
// int p(int I, int j, int k, int l) {   ; parameters i:r26 j:r27 k:r28, l:29
//    return min(min(g, i, j), k, l);
// }

// unoptimised

p:      add     r9, r0, r10     ; set up 1st parameter (g)
        add     r26, r0, r11    ; set up 2nd parameter (i)
        add     r27, r0, r12    ; set up 2nd parameter (j)
        callr   r25, min        ; call min (save return address in r25)
        xor     r0, r0, r0      ; nop in delay slot
        add     r1, r0, r10     ; set up 1st parameter (result returned from min in r1)
        add     r28, r0, r11    ; set up 2nd parameter (k)
        add     r29, r0. r12    ; set up 3rd parameter (l)
        callr   r25, min        ; call min (save return address in r25)
        xor     r0, r0, r0      ; nop in delay slot
        ret     r25, 0          ; return result in r1
        xor     r0, r0, r0      ; nop in delay slot
```

```
// optimised
//
p:      add     r9, r0, r10         ; set up 1st parameter (g)
        add     r26, r0, r11        ; set up 2nd parameter (i)
        callr   r25, min            ; call min (save return address in r25)
        add     r27, r0, r12        ; set up 3rd parameter (j) in delay slot
        add     r1, r0, r10         ; set up 1st parameter (result returned from min in r1)
        add     r28, r0, r11        ; set up 2nd parameter (k)
        callr   r25, min            ; call min (save return address in r25)
        add     r29, r0. r12        ; set up 3rd parameter (l) in delay slot
        ret     r25, 0              ; return result in r1
        xor     r0, r0, r0          ; nop in delay slot

//  int gcd(int a, int b) {         ; parameters a: r26 b:27
//      if (b == 0) {
//          return a;
//      } else {
//          return gcd(b, a % b); ; call an external function mod to evaluate a % b
//      }
//  }

// unoptimised

gcd:    sub     r27, r0, r0 {c} ; b == 0
        jne     gcd0            ; jump not equal
        xor     r0, r0, r0      ; nop in delay slot
        add     r26, r0, r1     ; set result (a)
        ret     r25, 0          ; return
        xor     r0, r0, r0      ; nop in delay slot
gcd0:   add     r26, r0, r10    ; set up 1st parameter (a)
        add     r27, r0, r10    ; set up 2nd parameter (b)
        callr   r25, mod        ; call mod(a, b)
        xor     r0, r0, r0      ; nop in delay slot
        add     r27, r0, r10    ; set up 1st parameter (b)
        add     r1, r0, r11     ; set up 2nd parameter (a % b)
        callr   r25, gcd        ; recursively call gcd(b, a % b)
        xor     r0, r0, r0      ; nop in delay slot
        ret     r25, 0          ; return
        xor     r0, r0, r0      ; nop in delay slot

// optimised

gcd:    sub     r27, r0, r0 {c} ; b == 0
        jeq     gcd0            ; jump equal
        add     r26, r0, r1     ; set result (a) can be executed if test true or false
        add     r26, r0, r10    ; set up 1st parameter (a)
        callr   r25, mod        ; call mod
        add     r27, r0, r10    ; set up 2nd parameter (b) in delay slot
        add     r27, r0, r10    ; set up 1st parameter (b)
        callr   r25, gcd        ; recursively call gcd(b, a % b)
        add     r1, r0, r11     ; set up 2nd parameter (a % b) in delay slot
gcd0:   ret     r25, 0          ; return
        xor     r0, r0, r0      ; nop in delay slot
```

Q2

Please also study the CS3021/3421 RISC I notes with the following description.

Code has been added to the ackermann function at entry and exit to simulate the on-chip register file overflow underflow mechanism. Figures 1 and 2 show an on-chip register file, with 6 register windows, positioned on top of a number of register windows already pushed onto a stack in memory.

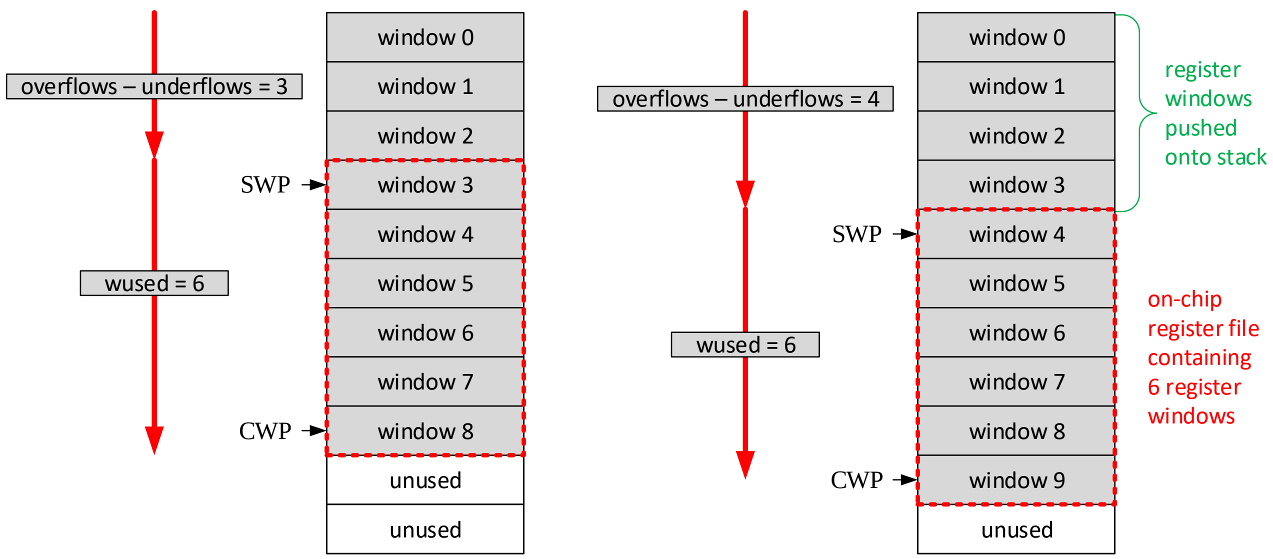

Fig.1 Register window overflow

Fig 1 illustrates register window overflow. There are 9 register windows in use: 3 (overflows – underflows) on the stack and 6 (wused) in the on-chip register file. CWP points to the current register window and SWP points to the oldest valid register window in the on-chip register file. An overflow occurs on a function call if CWP+1 == SWP (calculated using modulo arithmetic to simulate the circular register file). The register window pointed to by SWP is pushed on to the stack and CWP and SWP are incremented. If no overflow occurs, only CWP needs to be incremented. The circular nature of the on-chip register file is illustrated by the red rectangle, representing the on-chip register file, moving down one window slot on an overflow. Since there are 6 register windows, an alternative test for overflow is if wused == 6.

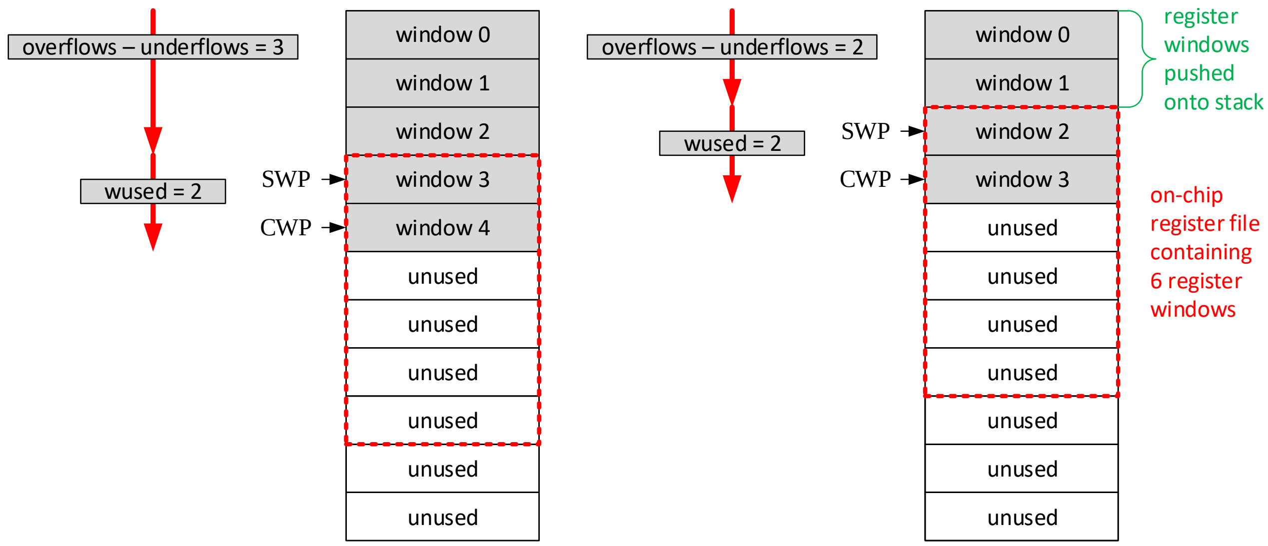

Fig. 2 Register window underflow

Fig.2 illustrates register underflow. There are 5 register windows in use: 3 (overflows – underflows) on the stack and 2 (wused) in the on-chip register file. CWP points to the current register window and SWP points to the oldest valid register window in the on-chip register file. There must always be two valid register windows in the on-chip register file due to the overlap between windows. CWP's r26..r31 are in SWP's register window. An underflow occurs on a function return if CWP-1 == SWP (calculated using modulo arithmetic to simulate the circular register file). SWP and CWP are decremented and a register window is popped from the stack and stored in the register window pointed to by the new SWP. If no underflow occurs, only CWP needs to be decremented. The circular nature of the on-chip register file is illustrated by the red rectangle, representing the on-chip register file, moving up one window slot on underflow. An alternative test for underflow is if wused == 2.

Method 0

At entry:

```
depth++;
if (depth > depthMax)
    depthMax = depth;

if (wused == nwindows)
    overflows++;
} else {
    wused++;
}
```

At exit:

```
depth--;

if (wused ==2) {
    underflows++;
} else {
    wused--;
}
```

Initialise depth = 0, depthMax = 0, overflows = 0, underflows = 0, wused = 2

Method 1

```
// defensive code to make sure % operator with a negative dividend works as expected

#define CIRCULAR(v, n) (n > 0 ? (v + n) % nwindows : (v + n + nwindows) % nwindows)
```

At entry:

```
depth++;
if (depth > depthMax)
    depthMax = depth;

if (CIRCULAR(cwp, 1) == swp)
    overflows++;
    swp = CIRCULAR(swp, 1)
}
cwp = CIRCULAR(cwp, 1);
```

At exit:

```
depth--;

if (CIRCULAR(cwp, -1) == swp) {
    underflows++;
    swp = CIRCULAR(swp, -1)
}
cwp = CIRCULAR(cwp, -1);
```

Initialise depth = 0; depthMax = 0, overflows = 0, underflows = 0, cwp = 0, swp = nwindows -1

Both methods also keep track of the current procedure depth and the maximum depth.

<u>Q3.</u>

See attached source code for Q3.

<u>24-Nov-2015</u>

ZINFANDEL Windows 7 Professional (64 bit) 64 bit exe NCPUS=8 RAM=64GB
Intel64 family 6 model 94 stepping 3 Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz

```
C:\Windows\system32\cmd.exe
ackerman(3, 6) = 509, calls = 172233, max depth = 511
nwindows = 6, overflows = 84885, underflows = 84885

ackerman(3, 6) = 509, calls = 172233, max depth = 511
nwindows = 8, overflows = 83911, underflows = 83911

ackerman(3, 6) = 509, calls = 172233, max depth = 511
nwindows = 16, overflows = 80142, underflows = 80142

starting...
ackermann(3, 6) = 509
execution time = 0.13ms
```

zinfandel Linux 4.2.0-18-generic 64 bit NCPUS=8 RAM=63GB
Intel64 family 6 model 94 stepping 3 Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz

```
Terminal File Edit View Search Terminal Help
jones@zinfandel:/mnt/windows/Courses/Software Examples/ackermann/Release$ ./ackermann
ackerman(3, 6) = 509, calls = 172233, max depth = 511
nwindows = 6, overflows = 84885, underflows = 84885

ackerman(3, 6) = 509, calls = 172233, max depth = 511
nwindows = 8, overflows = 83911, underflows = 83911

ackerman(3, 6) = 509, calls = 172233, max depth = 511
nwindows = 16, overflows = 80142, underflows = 80142

starting...
ackermann(3, 6) = 509
execution time = 0.08ms
jones@zinfandel:/mnt/windows/Courses/Software Examples/ackermann/Release$
```

NB: Linux version is almost twice as fast, which is somewhat surprising.