EN605.204.82.SU17 Computer Organization

Caleb Bowers

Assignment 9a

Original code:

```
(1) := #1 Indx
(2) BGT Indx #8 (20)
(3) - Indx #1 i1
(4) * i1 #10 i2
(5) * #5 DEF i3
(6) - i3 #1 i4
(7) - i4 #1 i5
(8) + i2 i5 i6
(9) * i6 #4 i7
(10) - Indx #1 i8
(11) * i8 #10 i9
(12) * #5 DEF i10
(13) - i10 #1 i11
(14) + i9 i11 i12
(15) * i12 #4 i13
(16) := Y[i13] X[i7]
(17) + #1 Indx i14
(18) := i14 Indx
(19) JMP (2)
(20)
```

Optimized Loop:

```
   (1)   :=    #0               Indx
   (2)   *     #5     DEF       i3
   (3)   -      i3     #2        i4
   (4)   BGT   Indx    #7        (12)
   (5)   *     Indx    #10       i1
   (6)   +     i1      i4        i5
   (7)   >>    i5      #2        i6
   (8)   +     i6      #4        i7
   (9)   :=    Y[i7]             X[i6]
   (10)  +     Indx    #1        i8
   (11)  :=    i8                Indx
   (12)  JMP                     (4)
   (13)
```

There were several optimization steps taken to achieve the “Optimized Loop” above. Initially, the original code can be broken into three blocks. Block One is Line (1), Block Two is Line (2) through Line (19), and Block Three is Line (20).

My first step was to initialize Indx to zero rather than 1, and reduce the Branch statement to branch if Indx is greater than 7. This removes lines (3) and (10). It also changes lines (4) and (11) to use Indx directly, rather than another register (i1 = Indx * 10). I then moved line (5) through line (7) and line (12) through line (13) outside of the loop. These are the same calculations. I also reduced it down to the new operations we see in the Optimized Loop on lines (2) through (3).

I focused next on the operations within the loop. Lines (5) through (8) of the Optimized Loop achieve what lines (8) through (9) and (14) through (15). This reduction came from algebra and was done to remove duplicate common sub expressions:

i3 = DEF * 5
i4 = i3 – 2
(inside loop):
i1 = Indx * 10
i5 = i1 + i4
i6 = i5 * 4

We then need to know i2 +(i4 + 1), so this gives us:
i1 + (i4 + 1) = i5 + 1
i7 = (i5 + 1) *4 = (i5 * 4) + 4 = i6 + 4
so from this we have line (8) in the optimized loop.

I have replaced the multiplication by 4 with a shift by two bytes, which is the same as multiplying by 4 and is faster.

I then assign the Y[i7] to X[i6] and increase Indx by one to move the iteration of the loop.