Team 04
Haohan Jiang, g3jiangh
Maria Yancheva, c2yanche
Timo Vink, c4vinkti
Chandeep Singh, g2singh

1 Program 1

As described above, we start with the `HALT` instruction at address 0, which will be used as our return address for the 'main procedure'.

```
HALT
```

Next we need to put the activation record on the stack and set the display register to point to it. The activation record contains the return address `0` (1 word), space to save a display register (1 word), and space for local variables (2683 words).

```
# Set display register
PUSHMT
SETD        0

# Create activation record
PUSH        0
PUSH        UNDEFINED
PUSH        2684
DUPN
```

The first line in the program that requires computation is line `1-4`. We need to evaluate the expression and store the result in the address of `k`. The addresses of `i, j, k, l` are 2, 3, 4, 5 from the activation record base respectively.

```
# Get address of k
ADDR        0       4

# Calculate (i + 3)
ADDR        0       2
LOAD
PUSH        3
ADD

# Calculate (j * k), subtract from the above
ADDR        0       3
LOAD
ADDR        0       4
MUL
SUB

# Calculate (k / l), add to the above
```

```
ADDR        0       4
LOAD
ADDR        0       5
DIV
ADD

# Store result in k
STORE
```

Next up, we have lines 1-6 and 1-7. We need to store constants in p and q, which are at offsets 7 and 8 from the activation record base respectively.

```
# Store TRUE in p
ADDR        0       7
PUSH        1
STORE

# Store FALSE in q
ADDR        0       8
PUSH        0
STORE
```

Next up we have line 1-8. We need to evaluate the expression and store the result in the address of r. The addresses of p, q, r, s are 7, 8, 9, 10 from the activation record base respectively. Recall that $s \land \neg p \equiv \neg(\neg s \lor p)$.

```
# Get address of r
ADDR        0       9

# Calculate (!q)
PUSH        MACHINE_TRUE
ADDR        0       8
LOAD
SUB

# Calculate (p | q), OR with result above
ADDR        0       7
LOAD
ADDR        0       8
LOAD
OR
OR

# Calculate (s & !p), OR with result above
PUSH        MACHINE_TRUE
PUSH        MACHINE_TRUE
ADDR        0       10
LOAD
SUB
```

```
PUSH        MACHINE_TRUE
PUSH        MACHINE_TRUE
ADDR        0       7
LOAD
SUB
SUB
OR
SUB
OR

# Store result in r
STORE
```

We're now at line 1-9. We need to evaluate the expression and store the result at the address of p. The addresses of `i, j, k, l, p` are 2,3,4,5,7 from the activation record base respectively. Recall that $a \leq b \equiv \neg(a > b)$.

```
# Get address of p
ADDR        0       9

# Calculate (i < j)
ADDR        0       2
LOAD
ADDR        0       3
LOAD
LT

# Calculate (k <= l), OR with result above
PUSH        MACHINE_TRUE
ADDR        0       5
LOAD
ADDR        0       4
LOAD
LT
SUB
OR

# Calculate (j = l), OR with result above
ADDR        0       3
LOAD
ADDR        0       5
LOAD
EQ
OR

# Store result in p
STORE
```

Similar to before, for line 1-19 we need to evaluate the expression and store the result at the address of `s` using the fact that $a \neq b \equiv \neg(a = b)$ as well as the two equivalences outlined for the previous two lines. The addresses of `j, k, m, r, s` are 3, 4, 6, 9, 10 from the activation record base respectively.

```
# Get address of s
ADDR        0       9

# Calculate !(k != m)
PUSH        MACHINE_TRUE
PUSH        MACHINE_TRUE
PUSH        MACHINE_TRUE
ADDR        0       4
LOAD
ADDR        0       6
LOAD
EQ
SUB
SUB

# Calculate !(j >= k), OR with result above and negate
PUSH        MACHINE_TRUE
ADDR        0       3
LOAD
ADDR        0       4
LOAD
LT
SUB
OR
SUB

# Calculate !(r = s), OR with result above
PUSH        MACHINE_TRUE
ADDR        0       9
LOAD
ADDR        0       10
LOAD
EQ
SUB
OR

# Store result in s
STORE
```

Next up is line 1-11. No new concepts here. The addresses of `q, r, s` are 8, 9, 10 from the activation record base respectively.

```
# Get address of q
```

```
ADDR        0       8

# Calculate (r = s)
ADDR        0       9
LOAD
ADDR        0       10
LOAD
EQ

# Calculate (!s != r), OR with result above
PUSH        MACHINE_TRUE
PUSH        MACHINE_TRUE
ADDR        0       10
LOAD
SUB
ADDR        0       9
LOAD
EQ
SUB
OR

# Store result in q
STORE
```

Next line requiring any computation is line 1-14. We know the stride of the first dimension of B is 151. The base addresses of A, B are 12, 19 from the activation record base respectively, and the offsets of i, j are 2, 3 respectively.

```
# Get base address of B
ADDR        0       19

# Calculate offset due to first dimension
ADDR        0       2
LOAD
PUSH        1
ADD
PUSH        -100
SUB
PUSH        151
MUL

# Calculate offset due to second dimension
ADDR        0       3
LOAD
PUSH        100
SUB
PUSH        -40
SUB
```

```
# Combine results to find address of B[i + 1, j - 100]
ADD
ADD

# Get value at A[j - 2]
ADDR        0       12
ADDR        0       3
LOAD
PUSH        2
SUB
PUSH        1
SUB
ADD
LOAD

# Store result in B[i + 1, j - 100]
STORE
```

And similarly for line 1-15. We know the stride of the first dimension of `D` is 50. The base addresses of `C, D` are 1680, 1685 from the activation record base respectively, and the offsets of `i, k` are 2, 4 respectively.

```
# Get address of C[-4]
ADDR        0       1680
PUSH        -4
PUSH        -7
SUB
ADD

# Get base address of D
ADDR        0       1685


# Calculate offset due to first dimension
ADDR        0       2
LOAD
PUSH        20
ADD
PUSH        -100
SUB
PUSH        50
MUL

# Calculate offset due to second dimension
ADDR        0       4
LOAD
PUSH        7
```

```
SUB
PUSH        1
SUB

# Combine results to find address of D[i + 20, k - 7]
ADD
ADD

# Store result in C[-4]
STORE
```

We're now at the end of the 'main procedure'. So we need to clean up the activation record and branch to the return address, which is where the `HALT` instruction is.

```
# Clean up activation record
PUSH        2684
POPN

# Branch to return address
ADDR        0       0
LOAD
BR
```