9.1 WCET Analysis
Determine the WCET of the following program which is written in pseudo code.

```plaintext
a = 1;
if (b >= 0) then {
    i = a + 2;
    n = (a + 3)^2 + 2;
    m = a + 3;
    b = 6;
} else {
    l = a + 6;
    n = l + 8;
    i = a + 4;
}
while (n >= i) do {
    if (b > 5) {
        a = 3;
        l = l + 1;
    } else {
        m = m + 1;
        a = 4;
    }
    i = i + 1;
}
```

The underlying machine executing the program is specified as follows:

- the processor has no pipeline
- the processor has no registers, i.e. all variables are stored in memory
• in an assignment, predicate evaluation, or arithmetic operation, data variables are accessed in order from right to left e.g.

\[ a = b + c \]

variables are accessed in the order \( c, b, a \)

• consider only the data cache

• reads and writes from/to a data variable are treated the same way. If the required data is in the cache, there is a cache HIT and it takes 1 cycle to read/write the data. If the data is not in the cache, there is a cache MISS and it takes 100 cycles to read/write the data

• execution model is very simplified. No temporary results are considered. Execution times of all operations (logical, mathematical, branching, and their combinations) on the processor are 1 cycle e.g.

\[ a = b*c + d/v^2; \]

takes 1 cycle to execute, NOT considering the loading and storing of data variables (the WCET of a statement will mainly depend on cache MISS and cache HIT of the data variables participating in the statement). Assignments such as \( a=2 \) depend entirely on whether the variable \( a \) is in cache or not.

• the data cache has only one cache set, i.e. it is fully associative. It can store 4 blocks of data. Each cache block can contain exactly one data variable

• the cache uses LRU replacement policy

Initial conditions: \( a, b, i, l, m, n \in (-\infty, \infty) \) and cache state is \((b, -, -, -)\), i.e. \( b \) is the most recently used variable and all other blocks are empty.

Tasks:

(a) Determine the basic blocks of the program

**Solution:**

See Figure 1.

(b) Determine the intervals for each variable at each program statement using static value analysis

**Solution:**

\[ a, b, i, l, m, n \in (-\infty, \infty) \]

**B1:**

\[ a = 1 \]

\[ a \in [1, 1] \]

\[ b, i, l, m, n \in (-\infty, \infty) \]

\[ b \geq 0 \]

\[ a \in [1, 1] \]

\[ b, i, l, m, n \in (-\infty, \infty) \]
B2:
\[ i = a + 2 \]
\[ i \in [3, 3] \]
\[ a \in [1, 1] \]
\[ b \in [0, \infty) \]
\[ l, m, n \in (-\infty, \infty) \]
\[ n = (a + 3)^2 + 2 \]
\[ n \in [18, 18] \]
\[ i \in [3, 3] \]
\[ a \in [1, 1] \]
\[ b \in [0, \infty) \]
\[ l, m \in (-\infty, \infty) \]
\[ m = a + 3 \]
\[ m \in [4, 4] \]
\[ n \in [18, 18] \]
\[ i \in [3, 3] \]
\[ a \in [1, 1] \]
\[ b \in [0, \infty) \]
\[ l \in (-\infty, \infty) \]
\[ b = 6 \]
\[ b \in [6, 6] \]
\[ m \in [4, 4] \]
\[ n \in [18, 18] \]
\[ i \in [3, 3] \]
\[ a \in [1, 1] \]
\[ l \in (-\infty, \infty) \]

B3:
\[ l = a + 6 \]
\[ l \in [7, 7] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 0) \]
\[ i, m, n \in (-\infty, \infty) \]
\[ n = l + 8 \]
\[ n \in [15, 15] \]
\[ l \in [7, 7] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 0) \]
\[ i, m \in (-\infty, \infty) \]
\[ i = a + 4 \]

\[ i \in [5, 5] \]
\[ n \in [15, 15] \]
\[ l \in [7, 7] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 0) \]
\[ m \in (-\infty, \infty) \]

Join of B2 and B3:

\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

First pass through the loop:

B4:
while (n >= i)

\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

B5:
if (b > 5)

\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

B6:
a = 3

\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [3, 3] \]
\[ b \in (5, 6] \]
\[ l, m \in (-\infty, \infty) \]

\[ l = l + 1 \]
\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [3, 3] \]
\[ b \in (5, 6] \]
\[ l, m \in (-\infty, \infty) \]

**B7:**
\[ m = m + 1 \]
\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 5] \]
\[ l, m \in (-\infty, \infty) \]
\[ a = 4 \]
\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [4, 4] \]
\[ b \in (-\infty, 5] \]
\[ l, m \in (-\infty, \infty) \]

**Join of B6 and B7:**
\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [3, 4] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

**B8:**
\[ i = i + 1 \]
\[ i \in [4, 6] \]
\[ n \in [15, 18] \]
\[ a \in [3, 4] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

**Join of B8, B2, and B3:**
\[ i \in [3, 6] \]
\[ n \in [15, 18] \]
\[ a \in [1, 4] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

If we do not have loop bounds, the loop will be traversed infinitely many times (or until a fixed point is reached). In this example, it means that the interval for variable \( i \) will increase with each iteration. Therefore the resulting intervals are as follow:
\[ i \in [3, \infty) \]
\[ n \in [15, 18] \]
\[ a \in [1, 4] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]

In such cases it is possible to include loop bounds analysis in the static value analysis or the programmer could be asked to supply loop bounds. This is actually the case for this simple program where we can see that the loop is executed at most 16 times.

(c) Determine the WCET for each basic block

**Solution:**

Use cache MUST analysis to determine the WCET of each basic block.

- \(H\) for hit, \(M\) for miss,
- \(c_i\) is WCET in cycles for the \(i\)-th basic block
- Initial cache state: \((b, -, -, -)\)

**B1:**
- \(a = 1\)
  - \(M\), \((a, b, -, -)\)
  - if \((b \geq 0)\)
    - \(H\), \((b, a, -, -)\)
  - \(c_1: 102\)

**B2:**
- \(i = a + 2\)
  - \(H\), \((a, b, -, -)\)
  - \(M\), \((i, a, b, -)\)
  - \(n = (a + 3)^2 + 2\)
    - \(H\), \((a, i, b, -)\)
    - \(M\), \((n, a, i, b)\)
  - \(m = a + 3\)
    - \(H\), \((a, n, i, b)\)
    - \(M\), \((m, a, n, i)\)
  - \(b = 6\)
    - \(M\), \((b, m, a, n)\)
  - \(c_2: 406\)

**B3:**
- \(l = a + 6\)
  - \(H\), \((a, b, -, -)\)
  - \(M\), \((l, a, b, -)\)
  - \(n = 1 + 8\)
    - \(H\), \((l, a, b, -)\)
    - \(M\), \((n, 1, a, b)\)
  - \(i = a + 4\)
    - \(H\), \((a, n, l, b)\)
    - \(M\), \((i, a, n, l)\)
  - \(c_3: 306\)
Join of B2 and B3: ({{},{}},{{a},{n}})

First pass through the loop
with initial cache state: ({{},{}},{{a},{n}})

B4:
while (n >= i)
    M, ({{i},{}},{{},{}},{{a}})
    M, ({{n},{}},{{i},{}},{{}})

B5:
if (b > 5)
    M, ({{b},{}},{{n},{}},{{i},{}},{{}})

B6:
a = 3
    M, ({{a},{}},{{b},{}},{{n},{}},{{i}})
    l = l + 1
    M, ({{l},{}},{{a},{}},{{b},{}},{{n}})
    H, ({{l},{}},{{a},{}},{{b},{}},{{n}})

B7:
m = m + 1
    M, ({{m},{}},{{b},{}},{{n},{}},{{i}})
    H, ({{m},{}},{{b},{}},{{n},{}},{{i}})
a = 4
    M, ({{a},{}},{{m},{}},{{b},{}},{{n}})

Join of B6 and B7: ({{},{}},{{a},{}},{{b},{}},{{n}})

B8:
i = i + 1
    M, ({{i},{}},{{},{}},{{a},{}},{{b}})
    H, ({{i},{}},{{},{}},{{a},{}},{{b}})

Join of B8, B2, and B3: ({{},{}},{{},{}},{{a},{}},{{}})

Second pass through the loop
with initial cache state: ({{},{}},{{},{}},{{a},{}},{{}})

B4:
while (n >= i)
    M, ({{i},{}},{{},{}},{{},{}},{{a}})
    M, ({{n},{}},{{i},{}},{{}},{}))
c4: 201

B5:
if (b > 5)
    M, ({{b},{}},{{n},{}},{{i},{}},{{}})
C5: 101

B6:
  a = 3
  M, ({a},{b},{n},{i})
  l = l + 1
  M, ({l},{a},{b},{n})
  H, ({l},{a},{b},{n})
c6: 202

B7:
  m = m + 1
  M, ({m},{b},{n},{i})
  H, ({m},{b},{n},{i})
  a = 4
  M, ({a},{m},{b},{n})
c7: 202

Join of B6 and B7: ({ },{a},{b},{n})

B8:
  i = i + 1
  M, ({i},{ },{a},{b})
  H, ({i},{ },{a},{b})
c8: 102

Join of B8, B2, and B3: ({ },{ },{a},{ })

Since ({ },{ },{a},{ }) equals the initial abstract cache state for the second cache analysis pass of the loop, it means we have found a fixed point (going through the loop again will not change the abstract state) and the loop does not need to be traversed anymore.

In this example, we can see that variable i is always pushed out of the abstract cache state (at the join of B8 with B2 and B3) even though it was the last variable accessed in the loop. This will make the analysis conclude wrongly that there always is a cache MISS when the statement while(n >= i) is executed even if the loop is executed several times. In such cases, if we are sure that the loop will be executed at least once, it will be better to unroll the loop once and then perform the analysis to get improved (smaller) WCET.

(d) Write down the ILP whose solution would determine the WCET of the whole program

**Solution:**

*Flow variables *d_i* are defined in Figure 1.*

*Structural constraints:*

\[ d_1 = d_2 + d_3 = x_1 \]
\[ d_2 = d_4 = x_2 \]
\[ d_3 = d_5 = x_3 \]
\[ d_4 + d_5 + d_{11} = d_6 + d_{12} = x_4 \]
\[d_6 = d_7 + d_8 = x_5\]
\[d_7 = d_9 = x_6\]
\[d_8 = d_{10} = x_7\]
\[d_9 + d_{10} = d_{11} = x_8\]

Assume program is executed once: \(d_1 = 1\).

Additional constraints:

- Loop bounds: considering the different paths in the program and variable values, loop is executed at most 16 times: \(x_5 \leq 16 * (x_2 + x_3)\).

Objective function:

\[
WCET = \max\{\sum_{i=1}^{8} c_i * x_i\}
\]
Figure 1: Basic blocks of the program