Aplikasi Modulo pada *Direct-Mapped Cache*

Faris Hasim Syauqi 13519050[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13519050@std.stei.itb.ac.id

***Abstract*—Cache adalah salah satu memori yang ada pada komputer yang berfungsi untuk meningkatkan kinerja komputer dengan cara berperan sebagai tempat penyimpanan sementara yang terletak lebih dekat dengan unit proses dibandingkan memori utama. Data akan disimpan pada cache di posisi tertentu sesuai dengan alamat yang menyimpan data tersebut dengan memanfaatkan operasi modulo yang memetakan alamat tersebut ke posisi pada cache. Pada makalah ini akan dibahas mengenai bagaimana pembacaan pada cache dilakukan dan bagaimana alamat dipetakan ke posisi pada cache dengan operasi modulo dengan menggunakan program simulasi cache yang dibuat dengan bahasa pemrograman C.**

***Keywords*—Cache, Data, Komputer, Modulo**

I. PENDAHULUAN

Dalam dunia komputer, memori adalah salah satu bagian terpenting. Hal ini dikarenakan untuk menyimpan informasi pada komputer diperlukan sebuah wadah. Di dalam komputer, memori adalah wadah untuk menyimpan berbagai informasi tersebut. Memori yang ada pada komputer diantaranya adalah memori utama, memori cache, dan memori sekunder.

Cache adalah salah satu memori pada komputer yang terletak di antara memori utama dan unit proses komputer (CPU) dan berperan penting sebagai mediator dan katalis. Ketika suatu proses sedang berlangsung pada komputer, dapat terjadi transfer data dari memori utama ke unit proses. Seringkali transfer data tersebut membutuhkan waktu yang jauh lebih lama dibandingkan proses yang sedang berlangsung itu sendiri. Oleh sebab itu, cache diciptakan sebagai salah satu cara untuk meningkatkan kinerja komputer dengan cara bertindak sebagai tempat penyimpan sementara dengan ukuran yang lebih kecil dibanding memori utama, namun pada posisi yang lebih dekat dengan unit proses.

Data yang disimpan pada cache akan dipetakan ke posisi tertentu pada cache sesuai dengan alamat memori utama yang menyimpan data tersebut. Proses pemetaan yang terjadi pada cache memanfaatkan operasi modulo. Dari alamat memori utama tempat data tersebut disimpan akan dilakukan proses modulo untuk menentukan di posisi manakah data tersebut akan disimpan pada cache.

II. LANDASAN TEORI

A. Bilangan Bulat

Bilangan bulat adalah bilangan yang tidak memiliki pecahan desimal, misalnya 0, 18, -300, 1234. Bilangan bulat berlawanan dengan bilangan riil yang memiliki pecahan desimal seperti 18.1, 1234.5, -3.0[1].

1. Sifat Pembagian pada Bilangan Bulat

Misalkan a dan b adalah dua buah bilangan bulat, dengan $a \neq 0$. Bilangan bulat a habis membagi b jika terdapat bilangan bulat c sedemikian sehingga $b = ac$.

Notasi : $a \mid b$ jika $b = ac$, $c \in Z$ dan $a \neq 0$. Salah satu contohnya adalah $4 \mid 12$ karena $12/4 = 3$ (bilangan bulat) atau $12 = 4 \times 3$. Tetapi $4 \nmid 13$ karena $13/4 = 3.25$ (bukan bilangan bulat) atau 4 tidak habis membagi 13.

2. Teorema Euclidean

Misalkan m dan n bilangan bulat, $n > 0$. Jika m dibagi dengan n maka hasil pembagiannya adalah q (quotient) dan sisanya r (remainder), sedemikian sehingga

$$m = nq + r,$$

dengan $0 \leq r < n$.

Contohnya bilangan 1987 dibagi 97 menghasilkan 20 dengan sisa 47, maka dapat dinyatakan sebagai

$$1987 = 20 \cdot 97 + 47.$$

dengan $q = 20$ dan $r = 47$.

Contoh lainnya untuk bilangan negatif adalah -22 dibagi 3 menghasilkan -8, dengan sisa 2, sehingga dapat dinyatakan sebagai

$$-22 = (-8) \cdot 3 + 2.$$

dengan $q = -8$ dan $r = 2$.

3. Pembagi Bersama Terbesar (PBB)

Misalkan a dan b bilangan bulat tidak nol, Maka pembagi bersama terbesar (atau greatest common divisor (GCD)) dari a dan b adalah bilangan bulat terbesar d sedemikian hingga $d \mid a$ dan $d \mid b$. Dapat dinyatakan juga sebagai PBB(a, b) = d.

Contoh dua buah bialngan yaitu 45 dan 36. Bilangan 45 memiliki faktor pembagi yaitu 1, 3, 5, 9, 15, 45. Faktor pembagi bilangan 36 adalah 1, 2, 3, 4, 9, 12, 18, 36. Keduanya memilki faktor bersama yaitu 1,3,9. Karena faktor pembagi terbesarnya adalah 9, maka PBB(45,36) = 9.

B. Aritmatika Modulo

1. Definisi

Misalkan a dan m bilangan bulat (m > 0). Operasi a mod m (dibaca "a modulo m") memberikan sisa jika a dibagi dengan m. Notasi:

$$a \bmod m = r$$

sedemikian sehingga a = mq + r, dengan $0 \le r < m$. Bilangan m disebut modulus atau modulo, dan hasil aritmetika modulo m terletak di dalam himpunan {0, 1, 2, …, m – 1}.

Contoh 23 dibagi 5 menghasilkan sisa pembagian 3, sehingga dapat dinyatakan 23 mod 5 = 3. Kemudian 27 habis dibagi oleh 3, sehingga dapat dinyatakan 27 mod 3 = 0. Contoh lainnya yaitu -41 dibagi 9 memberikan sisa pembagian 4, sehingga dapat dinyatakan -41 mod 9 = 4.

2. Kongruen

Misalkan a dan b bilangan bulat dan m adalah bilangan > 0, maka

$$a \equiv b \pmod m$$

jika dan hanya jika m | (a – b). Jika a tidak kongruen dengan b dalam modulus m, maka ditulis $a \not\equiv b \pmod m$.

Kekongruenan pada dua buah bilangan a dan b dalam modulus suatu bilangan m menyatakan bahwa a dan b memberikan hasil pembagian yang sama ketika dibagi dengan bilangan positif m, yaitu

$$a \bmod m = b \bmod m,$$
$$a = b + km$$

dengan k sembarang bilangan bulat. Contohnya yaitu 38 mod 5 = 23 mod 5 = 3, maka 38 kongruen dengan 23 dalam modulus 3, atau $38 \equiv 23 \pmod 5$.

Selain itu, notasi operasi modulo juga dapat dinyatakan dengan kekongruenan, a mod m = r dapat dinyatakan dengan $a \equiv r \pmod m$. Misalnya 23 mod 5 = 3, sehingga $23 \equiv 3 \pmod 5$.

3. Teorema Sifat-sifat Modulo

Misalkan m adalah bilangan bulat positif. Jika $a \equiv b \pmod m$ dan c adalah sembarang bilangan bulat, maka

i. $(a + c) \equiv (b + c) \pmod m$
ii. $ac \equiv bc \pmod m$
iii. $a^p \equiv b^p \pmod m$, p bilangan bulat tak-negatif

Jika $a \equiv b \pmod m$ dan $c \equiv d \pmod m$, maka

i. $(a + c) \equiv (b + d) \pmod m$
ii. $ac \equiv bd \pmod m$

Contohnya misalkan $17 \equiv 2 \pmod 3$ dan $10 \equiv 4 \pmod 3$. Dengan menggunakan sifat-sifat modulo, karena $17 + 5 \equiv 2 + 5 \pmod 3$, maka $22 \equiv 7 \pmod 3$. Karena $17 \cdot 5 \equiv 2 \cdot 5 \pmod 3$ maka $85 \equiv 10 \pmod 3$. Karena $17 + 10 \equiv 2 + 4 \pmod 3$, maka $27 \equiv 6 \pmod 3$. Karena $17 \cdot 10 \equiv 2 \cdot 4 \pmod 3$, maka $170 \equiv 8 \pmod 3$.

C. Cache

1. Definisi

Cache adalah salah satu memori dalam komputer yang tercepat dalam pengaksesannya dan berperan sebagai mediator antara CPU dan Memori(RAM)[2]. Ketika CPU membutuhkan suatu data yang tersimpan maka data tersebut akan dicari terlebih dahulu pada cache dan jika tidak ditemukan pada cache maka selanjutnya akan dicari pada memori kemudian disimpan pada cache.

Ketika suatu proses sedang berjalan pada komputer, akan terjadi transfer data yang dibutuhkan proses tersebut. Transfer data yang terjadi dari memori utama ke unit proses membutuhkan waktu yang lama, bahkkan jauh lebih lama relatif terhadap proses itu sendiri. Cache yang bertindak sebagai mediator dapat meningkatkan kinerja komputer dengan cara mengurangi terjadinya transfer data dari memori utama ke unit proses.

Data yang dtransfer dari memori utama ke unit proses akan disimpan pada cache yang letaknya dekat dengan unit proses. Ketika data tersebut dibutuhkan kembali pada proses selanjutnya data cukup diambil dari cache, tidak perlu ada transfer lagi dari memori utama.

2. Terminologi

Ada beberapa terminologi penting pada cache. *Cache block* adalah unit dasar pada penyimpanan data di dalam cache. Sebuah blok dapat menampung lebih dari satu data. *Cache line* adalah unit dasar mirip seperti blok cache. *Cache set* adalah sebuah 'baris' pada cache. Banyaknya blok setiap set tergantung pada struktur cache yang digunakan. *Tag* adalah untuk identifikasi suatu kelompok data dengan kelompok data yang lain yang diletakkan pada set yang sama. *Valid bit* adalah sebuah bit yang menyatakan informasi pada blok valid atau tidak (umumnya menggunakan 0 atau 1)[3].

Cache hit terjadi ketika data yang diperlukan ada pada cache. *Cache miss* terjadi ketika data yang diperlukan tidak ada pada cache.

3. Struktur Cache

Set	Valid	Tag	Block
0	_	_	___
1	_	_	___
...	_	_	___
N-1	_	_	___

Gambar 1. Struktur cache dalam bentuk tabel (Sumber:Penulis)

a) *Fully Associative*
Pada cache tipe ini, sebuah data baru yang akan disimpan pada cache dapat diletakan pada blok manapun.

b) *Direct Mapped*
Pada cache tipe ini, data akan disimpan pada set tertentu tergantung dari alamat yang menyimpan data tersebut. Cache yang akan dibahas lebih lanjut pada makalah ini adalah cache dengan tipe ini.

c) *E-Way Associative Set*
Pada cache tipe ini, setiap set terdiri dari E-buah baris. Sama seperti Direct-mapped cache, Data akan disimpan pada set tertentu. Pada set tersebut data dapat disimpan pada baris manapun yang kosong atau baris yang terakhir kali diproses (*least recently used*). Pada dasarnya, Direct-mapped cache adalah E-way associative cache dengan E = 1.

4. Alokasi Data Pada Cache

Data akan disimpan pada cache di set tertentu berdasarkan alamat memori tempat menyimpan data tersebut. Misalkan N adalah banyaknya set pada cache dan B adalah ukuran setiap blok pada cache, maka set S tempat menyimpan data tersebut dapat dinyatakan sebagai fungsi dari alamat a yaitu

$$S(a) = \left\lfloor \frac{a}{B} \right\rfloor \ mod\ N$$

Selain itu, nilai tag T untuk identifikasi juga dapat dinyatakan sebagai fungsi dari alamat a yaitu

$$T(a) = \left\lfloor \frac{a}{B \times N} \right\rfloor$$

Setelah didapatkan nilat set dan tag, data dapat diidentifikasi apakah terjadi hit atau miss. Selain itu *miss-rate* pada cache dapat dihitung sebagai

$$Rate = \frac{Nmiss}{N} \times 100\%$$

Dengan N adalah banyaknya pembacaan pada cache dan Nmiss adalah banyaknya data yang miss ketika pembacaan.

5. Algoritma Cara Kerja Cache

Diberikan sebuah alamat memori utama dari suatu data. Dari alamat tersebut, akan dilakukan proses pada cache sesuai dengan algoritma berikut.

1) Dari alamat tersebut, dapat ditentukan set dimana data dengan alamat tersebut akan disimpan pada cache.
2) Setelah didapatkan indeks set, periksa indikator valid pada set tersebut, jika valid bit bernilai 0, maka data tidak ada pada cache, dengan kata lain miss. Jika valid bernilai 1, lanjut ke pemeriksaan selanjutnya
3) Setelah diperiksa valid bit, periksa tag yang diperoleh dari alamat dengan indikator tag pada set tersebut. Jika hasilnya cocok maka data ada pada cache, dengan kata lain hit. Jika tidak cocok, maka data tidak ada pada cache, dengan kata lain miss.
4) Jika terjadi miss, maka data akan diambil dari memori utama dan disimpan di dalam cache. Banyaknya data yang ditransfer dari memori utama tidak hanya satu data melainkan juga beserta data-data yang berada di dekatnya yang memilki nilai tag yang sama. Besarnya data yang ditransfer tergantung dari ukuran blok cache. Data tersebut disimpan dalam cache dengan index set yang sesuai dan indikator tag akan disimpan dengan nilai tag yang diperoleh dari alamat.

III. HASIL DAN PEMBAHASAN

A. *Source Code Program*

Berikut ini source code program untuk simulasi cache berdasarkan algoritma yang telah dibahas sebelumnya. Secara garis besar program yang dibuat terdiri dari struktur data, implementasi fungsi-fungsi, dan program utama.

1. Struktur data

```
#define NSET 3
#define NBLOK 5

//**** Struktur Data ****//
typedef struct tset
{
    boolean valid;
    int tag;
    int blok[NBLOK];
} SET ;

typedef SET Cache[NSET];

//**** KONSTRUKTOR ****//
void MakeEmpty(Cache * C);

//**** SELEKTOR ****//
#define Valid(S) S.valid
#define Tag(S) S.tag
#define Set(C,i) (C)[i]
#define Blok(S,i) S.blok[i]
```

Gambar 2. Struktur data program (Sumber : Penulis)

Struktur data yang digunakan pada percobaan ini terdiri dari dua buah tipe data. Tipe data yang pertama adalah SET yang merepresentasikan satu set pada cache. Setiap satu SET memilki komponen valid, tag dan blok. Tipe data yang kedua adalah Cache yang didefinisikan sebagai larik daripada SET. Ukuran Cache yang digunakan untuk simulasi ini adalah 3 Set blok dengan ukuran setiap blok adalah 5 alamat. Di dalam percobaan ini, data yang disimpan dan ditampilkan dalam cache adalah alamatnya.

2. Fungsi-fungsi

```
//**** FUNGSI-FUNGSI ****//
void InputCache(Cache * C, int address){
    // prekondisi : miss
    // KAMUS LOKAL
    int i, first, idxset, tag;

    // ALGORITMA
    first = (address / NBLOK) * NBLOK;
    idxset = FindSet(address);
    tag = FindTag(address);

    Valid(Set(*C,idxset)) = 1;
    Tag(Set(*C,idxset)) = tag;

    for(i = 0; i < NBLOK; i++) {
        Blok(Set(*C,idxset), i) = first + i;
    }
}
```

```c
int FindSet(int address) {
    // KAMUS
    // ALGORITMA
    return address/NBLOK % NSET;
}

int FindTag(int address) {
    // KAMUS
    // ALGORITMA
    return address/(NBLOK * NSET);
}

boolean Read(Cache C, int address) {
    // true jika hit
    // KAMUS
    int set, tag;

    // ALGORITMA
    set = FindSet(address);
    tag = FindTag(address);

    return Valid(Set(C, set)) && (Tag(Set(C, set)) == tag);
}
```

Gambar 3. Implementasi fungsi-fungsi program (Sumber : Penulis)

Fungsi-fungsi yang dibuat dalam program ini diantaranya fungsi findset dan findtag yang secara berturut-turut merupakan implementasi dari fungsi S(a) dan T(a). kemudian fungsi read untuk pembacaan pada cache.

3. Program Utama

```c
#include "cache.h"
#include <stdio.h>

int main() {
    // KAMUS
    Cache C;
    int i;
    int query;
    int N;

    // ALGORITMA
    MakeEmpty(&C);

    printf("\n");
    PrintCache(C);
    printf("\n");

    printf("Masukkan banyak query: ");
    scanf("%d", &N);
    for(i = 0; i < N; i++) {
        printf("Masukkan alamat memori: ");
        scanf("%d", &query);
        if (Read(C,query)){
            printf("\n// HIT //\n\n");
        } else {
            printf("\n// Miss //\n\n");
            InputCache(&C,query);
        }
        PrintCache(C);
        printf("\n");
    }
    return 0;
}
```

Gambar 4. Source code program utama (Sumber : Penulis)

B. *Hasil Percobaan*

Pada Percobaan ini, program simulasi cache akan dijalankan dan dilakukan percobaan dengan 5 data uji yaitu 23, 61, 28, 97 dan 25.

Pada keadaan awal dimulainya simulasi, cache akan di-inisiasi dalam keadaan 'dingin' yaitu tidak mengandung data apapun yang valid. Keadaan cache dapat dilihat pada gambar berikut.

```
SET [ valid | tag |       Blok       ]
 0  [   0   |  0  | [72,0,1531104,0,1] ]
 1  [   0   |  -1  | [-1,4201205,0,1,0] ]
 2  [   0   |  0  | [0,0,72,0,0] ]

Masukkan banyak query: 5
Masukkan alamat memori: 23
```

Gambar 5. Kondisi awal cache (Sumber : Penulis)

Pertama, program akan diberi masukan berupa banyaknya data uji (query) dan data uji yang pertama yaitu alamat 23. Untuk alamat a = 23, dengan banyaknya set N = 3 dan ukuran blok B = 5 dapat ditentukan nilai set-nya yaitu

$$S(23) = \left\lfloor \frac{23}{5} \right\rfloor mod\ 3\ = 4\ mod\ 3 = 1$$

Setelah diperoleh nilai set dimana data dengan alamat tersebut akan disimpan, akan dilakukan pemeriksaan pada set tersebut. Pada gambar dapat dilihat bahwa valid bit pada set 1 bernilai 0 mengakibatkan alamat 23 tidak ditemukan pada cache sehingga ditampilkan pesan miss dan cache pada set 1 akan disimpan dengan alamat 20 hingga 24. Data yang disimpan sebanyak 5 buah alamat sesuai dengan ukuran setiap blok. Selain itu, nilai valid akan disimpan dengan nilai 1 dan nilat tag juga akan disimpan dengan nilai tag milik alamat 23, yaitu

$$T(23) = \left\lfloor \frac{23}{5 \times 3} \right\rfloor = 1$$

Hasil eksekusi percobaan pertama dapat dilihat pada gambar di bawah ini.

```
Masukkan alamat memori: 23

// Miss //

SET [ valid | tag |       Blok       ]
 0  [   0   |  0  | [72,0,1531104,0,1] ]
 1  [   1   |  1  | [20,21,22,23,24] ]
 2  [   0   |  0  | [0,0,72,0,0] ]
```

Gambar 6. Hasil percobaan pertama, dengan input alamat 23 (Sumber : Penulis)

Selanjutnya pada percobaan kedua, alamat yang dimasukkan adalah 61, Nilai set dan tag dari alamat ini dapat ditentukan, yaitu

$$S(61) = \left\lfloor \frac{61}{5} \right\rfloor mod\ 3\ = 12\ mod\ 3 = 0$$

$$T(61) = \left\lfloor \frac{61}{5 \times 3} \right\rfloor = 4$$

Sama seperti percobaan sebelumnya, pada set dimana alamat 61 akan ditempatkan, dalam hal ini set ke-0, tidak ditemukan adanya data alamat 61 karena valid bit bernilai 0. Sehingga ditampilkan miss dan data pada cache diperbarui. Hasil eksekusi percobaan kedua dapat dilihat pada gambar berikut.

```
Masukkan alamat memori: 61

// Miss //

SET [ valid | tag |       Blok       ]
 0  [   1   |  4  | [60,61,62,63,64] ]
 1  [   1   |  1  | [20,21,22,23,24] ]
 2  [   0   |  0  | [0,0,72,0,0] ]
```

Gambar 7. Hasil percobaan kedua, dengan input alamat 61 (Sumber : Penulis)

Selanjutnya pada percobaan ketiga, alamat yang dimasukkan adalah 28, Nilai set dan tag dari alamat ini dapat ditentukan, yaitu

$$S(28) = \left\lfloor \frac{28}{5} \right\rfloor \ mod\ 3 \ = 5\ mod\ 3 = 2$$

$$T(28) = \left\lfloor \frac{28}{5 \times 3} \right\rfloor = 1$$

Sama seperti dua percobaan sebelumnya, pada set dimana alamat 28 akan ditempatkan, dalam hal ini set ke-2, tidak ditemukan adanya data alamat 28 karena valid bit bernilai 0. Sehingga ditampilkan miss dan data pada cache diperbarui. Hasil eksekusi percobaan ketiga dapat dilihat pada gambar berikut

```
Masukkan alamat memori: 28

// Miss //

SET [ valid | tag |       Blok       ]
 0  [   1   |  4  | [60,61,62,63,64] ]
 1  [   1   |  1  | [20,21,22,23,24] ]
 2  [   1   |  1  | [25,26,27,28,29] ]
```

Gambar 8. Hasil percobaan ketiga, dengan input alamat 28 (Sumber : Penulis)

Selanjutnya pada percobaan keempat, alamat yang dimasukkan adalah 97, Nilai set dari alamat ini dapat ditentukan, yaitu

$$S(97) = \left\lfloor \frac{97}{5} \right\rfloor \ mod\ 3 \ = 19\ mod\ 3 = 1$$

Berbeda dengan percobaan-percobaan sebelumnya, pada percobaan keempat ini, set dimana alamat 97 akan disimpan, dalam hal ini set ke-1, sudah mengandung data yang valid, dicirikan dengan valid bit pada set tersebut yang bernilai 1. Sehingga selanjutnya akan dilakukan pebandingan antara nilai tag yang dimiliki oleh alamat 97 dengan nilai tag yang ada pada cache. Nilai tag yang dimiliki oleh 97 adalah

$$T(97) = \left\lfloor \frac{97}{5 \times 3} \right\rfloor = 6 \ \neq 1$$

Karena nilai tag-nya berbeda dengan nilai yang ada pada cache pada set 1, maka disimpulkan alamat 97 tidak ada pada cache. Dengan kata lain terjadi miss dan cache akan diperbaharui khususnya pada set 1. Blok nya akan diisi dengan alamat 97 dan sekitarnya (yang sesuai dengan ukuran blok). Kemudian tag pada cache akan diubah menjadi nilai tag dari alamat 97, yaitu 6. Hasil eksekusi percobaan keempat dapat dilihat pada gambar berikut.

```
Masukkan alamat memori: 97

// Miss //

SET [ valid | tag |       Blok       ]
 0  [   1   |  4  | [60,61,62,63,64] ]
 1  [   1   |  6  | [95,96,97,98,99] ]
 2  [   1   |  1  | [25,26,27,28,29] ]
```

Gambar 9. Hasil percobaan keempat, dengan input alamat 97 (Sumber : Penulis)

Selanjutnya pada percobaan kelima, alamat yang dimasukkan adalah 25, Nilai set dari alamat ini dapat ditentukan, yaitu

$$S(25) = \left\lfloor \frac{25}{5} \right\rfloor \ mod\ 3 \ = 5\ mod\ 3 = 2$$

Pada percobaan kelima ini, set dimana alamat 25 akan disimpan, dalam hal ini set ke-2, sudah mengandung data yang valid, dicirikan dengan valid bit pada set tersebut yang bernilai 1. Sama seperti percobaan keempat, selanjutnya akan dilakukan pebandingan antara nilai tag dari alamat 25 dengan nilai tag yang ada pada cache. Nilai tag dari alamat 25 adalah

$$T(25) = \left\lfloor \frac{25}{5 \times 3} \right\rfloor = 1$$

Karena nilai tag dari alamat 25 sama dengan nilai tag yang sudah terdapat pada cache di set ke-2, maka dapat disimpulkan bahwa data yang berada di alamat 25 sudah tersimpan pada cache. Dengan kata lain pada kasus ini terjadi hit dan tidak akan ada perubahan pada cache. Cache tetap seperti keadaan sebelumnya. Hasil eksekusi percobaan kelima dapat dilihat pada gambar berikut.

```
Masukkan alamat memori: 25

// HIT //

SET [ valid | tag |       Blok       ]
 0  [   1   |  4  | [60,61,62,63,64] ]
 1  [   1   |  6  | [95,96,97,98,99] ]
 2  [   1   |  1  | [25,26,27,28,29] ]
```

Gambar 10. Hasil percobaan kelima, dengan input alamat 25 (Sumber : Penulis)

Setelah dilakukan semua percobaan tersebut, diperoleh data hasil percobaan berupa nilai set dan nilai tag dari alamat tersebut, serta indikator terjadinya hit atau miss. Data tersebut penulis sajikan dalam tabel berikut.

No	Alamat	Indeks Set	Tag	Hit/Miss
1	23	1	1	Miss
2	61	0	4	Miss
3	28	2	1	Miss
4	97	1	6	Miss
5	25	2	1	Hit

Tabel 1. Data hasil percobaan simulasi cache (Sumber:Penulis)

Dari data hasil percobaan di atas dapat ditentukan *miss-rate* dari seluruh percobaan yang dilakukan, yaitu

$$Rate = \frac{Nmiss}{N} \times 100\% = \frac{4}{5} \times 100\% = 80\%$$

Hasil *miss-rate* yang diperoleh cukup besar yaitu 80%, hal ini diakibatkan karena alamat yang dibaca pada setiap percobaan berjarak cukup jauh antara yang satu dengan yang lainnya, misalnya 23 pada percobaan pertama, kemudian 61 pada percobaan kedua, kemudian 97 pada percobaan ke 4. Selain itu juga adanya faktor-faktor yang lainnya seperti keadaan awal cache yang diinisiasi dengan cache 'dingin'. Ukuran cache juga dapat mempengaruhi *miss-rate* pada cache. Semakin besar ukuran cache maka *miss-rate* akan semakin kecil.

Dari percobaan yang telah dilakukan dapat dinyatakan bahwa cache bekerja dengan cara memanfaatkan operasi modulo. Data disusun dalam cache berdasarkan alamat memori yang menyimpan data tersebut. Alamat tersebut berperan sebagai peubah untuk fungsi modulo. Fungsi modulo memetakan alamat ke indeks set dimana data alamat tersebut akan disimpan pada cache.

IV. KESIMPULAN

Konsep teori bilang dalam ilmu matematika diskrit memiliki banyak penerapan pada berbagai bidang, salah satunya yaitu operasi aritmatika modulo yang diaplikasikan pada bidang ilmu komputer yaitu pada proses pembacaan dan pemetaan pada cache. Data akan disimpan pada cache sesuai dengan alamat memori yang menyimpan data tersebut. Dari alamat tersebut akan dilakukan operasi modulo untuk menentukan di posisi manakah data tersebut akan disimpan pada cache.

V. UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, penulis dapat menyelesaikan makalah dengan judul "Aplikasi Modulo pada *Direct-Mapped Cache*". Penulis juga menghaturkan ucapan terima kasih kepada berbagai pihak yang sudah membantu dalam penulisan makalah ini, kepada Dra. Harlili M.Sc, selaku dosen pengampu mata kuliah IF2120 Matematika Diskrit kelas K2 semester 1 tahun ajaran 2020/2021, kepada Dr. Ir. Rinaldi Munir, MT. yang telah menyediakan situs yang bermanfaat dan membantu dalam penyusunan makalah ini, juga kepada pihak-pihak lainnya yang tidak bisa disebutkan satu-persatu.

REFERENSI

[1] Munir, Rinaldi, "Teori Bilangan," vol. 1, no. Bagian 1, pp. 1–4, 1991.
[2] A. Khan, "Brief Overview of Cache Memory," no. April, 2020.
[3] S. Garcia, "The Basics of Caches," pp. 1–3.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2020

Faris Hasim Syauqi
13519050