Bab 11

Memory Layout dan Addressing Modes

Sub-CPMK yang Dicakup dalam Bab Ini:

- **Sub-CPMK 5.2:** Mengimplementasikan addressing modes untuk variabel dan arrays

11.1 Metode Pengalamatan (Addressing Modes)

Addressing Modes menentukan bagaimana operan diambil dari memori atau register pada tingkat bahasa mesin. Intermediate Code Generation bertindak sebagai jembatan antara front-end yang dependen pada bahasa sumber dan back-end yang dependen pada mesin target [1].

11.1.1 Jenis Pengalamatan Umum

- **Register Addressing**: Operan berada langsung di register (`ADD R1, R2`).
- **Immediate Addressing**: Operan adalah konstanta yang tertanam dalam instruksi (`ADDI R1, R1, 10`).
- **Displacement/Indexed**: Mengakses memori dengan alamat *base register* ditambah *offset* (`LW R1, 8(R2)`). Sangat berguna untuk *stack frame* dan akses *struct*.

11.1.2 Kalkulasi Alamat Array

Akses array `A[i]` diterjemahkan menjadi alamat:

$$\text{Alamat} = \text{Base}(A) + (i \times \text{ukuran\_elemen})$$

Kompilator harus menghasilkan instruksi perkalian dan penambahan untuk menghitung offset ini di setiap akses array.

11.2 Addressing Modes

11.2.1 Direct Addressing

- **Absolute**: Alamat memori langsung
- **Register**: Variabel dalam register
- **Immediate**: Konstanta dalam instruksi

```
// Direct addressing examples
int x = 10;           // x di memory dengan alamat spesifik
register int y = 20;  // y di register
#define Z 30          // Z sebagai immediate value
```

11.2.2 Indirect Addressing

- **Register indirect**: Alamat di register
- **Memory indirect**: Alamat di memory
- **Indexed**: Base + index

```
// Indirect addressing examples
int *ptr = &x;        // ptr menyimpan alamat x
int value = *ptr;     // Indirect melalui ptr
int arr[10];
int elem = arr[i];    // Indexed addressing
```

11.3 Array Addressing

11.3.1 One-Dimensional Arrays

Layout memory untuk 1D array:

```
int arr[5] = {10, 20, 30, 40, 50};

// Memory layout (each int = 4 bytes)
// arr[0] at base_address + 0*4
// arr[1] at base_address + 1*4
// arr[2] at base_address + 2*4
// arr[3] at base_address + 3*4
// arr[4] at base_address + 4*4

// Address calculation
int address = base_address + index * sizeof(int);
```

11.3.2 Multi-Dimensional Arrays

Row-major order untuk 2D arrays:

```c
int matrix[3][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};

// Address calculation: base + (row * cols + col) * sizeof(int)
int element_address = base_address +
                      (row * 4 + col) * sizeof(int);

// matrix[1][2] = base + (1*4 + 2)*4 = base + 6*4
```

11.3.3 Array Implementation

```c
typedef struct {
    void *base_address;
    int element_size;
    int num_dimensions;
    int *dimensions;
    int *bounds;  // lower bounds
} ArrayDescriptor;

int calculate_address(ArrayDescriptor *arr, int *indices) {
    int offset = 0;
    int stride = 1;

    // Calculate offset from last dimension to first
    for (int i = arr->num_dimensions - 1; i >= 0; i--) {
        offset += (indices[i] - arr->bounds[i]) * stride;
        stride *= arr->dimensions[i];
    }

    return (int)arr->base_address + offset * arr->element_size;
}
```

11.4 Structure and Union Layout

11.4.1 Structure Memory Layout

```c
struct Student {
    char name[50];      // 50 bytes
    int age;            // 4 bytes (aligned to 4)
    double gpa;         // 8 bytes (aligned to 8)
    char grade;         // 1 byte
    // Padding: 3 bytes for alignment
};

```

```
// Total size: 50 + 4 + 4(padding) + 8 + 1 + 3(padding) = 70 bytes
```

11.4.2 Union Memory Layout

```
union Data {
    int i;              // 4 bytes
    float f;            // 4 bytes
    char c[4];          // 4 bytes
    double d;           // 8 bytes (largest member)
};

// Union size = size of largest member = 8 bytes
```

11.5 Pointer Arithmetic

11.5.1 Pointer Operations

```
int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int *ptr = arr;

// Pointer arithmetic
ptr++;        // ptr += sizeof(int) (moves to next element)
ptr += 3;     // ptr += 3 * sizeof(int)
ptr--;        // ptr -= sizeof(int)

// Array access via pointers
int value = *(ptr + 2);  // Same as ptr[2]
int diff = ptr - arr;    // Number of elements between pointers
```

11.5.2 Dynamic Arrays

```
typedef struct {
    int *data;
    int size;
    int capacity;
} DynamicArray;

DynamicArray* create_array(int initial_capacity) {
    DynamicArray *arr = malloc(sizeof(DynamicArray));
    arr->data = malloc(initial_capacity * sizeof(int));
    arr->size = 0;
    arr->capacity = initial_capacity;
    return arr;
}

void resize_array(DynamicArray *arr, int new_capacity) {
    arr->data = realloc(arr->data, new_capacity * sizeof(int));
    arr->capacity = new_capacity;
}
```

11.6 Address Translation

11.6.1 Virtual to Physical Address

```c
typedef struct {
    unsigned int page_number : 10;
    unsigned int offset : 12;
} VirtualAddress;

typedef struct {
    unsigned int frame_number : 10;
    unsigned int valid : 1;
    unsigned int protection : 2;
} PageTableEntry;

unsigned int virtual_to_physical(VirtualAddress va,
                                 PageTableEntry *page_table) {
    PageTableEntry entry = page_table[va.page_number];
    if (entry.valid) {
        return (entry.frame_number << 12) | va.offset;
    }
    return 0;  // Page fault
}
```

11.6.2 Segmentation

```c
typedef struct {
    unsigned int base;
    unsigned int limit;
    int protection;
} SegmentDescriptor;

unsigned int segmented_address(unsigned int logical_addr,
                               SegmentDescriptor *segment) {
    if (logical_addr <= segment->limit) {
        return segment->base + logical_addr;
    }
    return 0;  // Segmentation fault
}
```

Aktivitas Pembelajaran

1. **Memory Layout**: Implementasikan simulator memory layout untuk berbagai tipe data.

2. **Array Addressing**: Bangun array descriptor untuk multi-dimensional arrays.

3. **Pointer Arithmetic**: Implementasikan pointer arithmetic operations.

4. **Structure Layout**: Analisis memory layout untuk structures dengan padding.

5. **Address Translation**: Implementasikan simple virtual memory translation.

Latihan dan Refleksi

1. Hitung memory layout untuk struktur dengan nested structures!

2. Implementasikan address calculation untuk 3D array dengan arbitrary bounds!

3. Analisis overhead dari different addressing modes!

4. Desain memory layout untuk object-oriented language dengan inheritance!

5. Implementasikan garbage collection-aware memory management!

6. **Refleksi**: Bagaimana memory layout mempengaruhi performance program?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 5.2

A. Pilihan Ganda

1. Row-major order untuk 2D array menghitung:
   (a) row * cols + col
   (b) col * rows + row
   (c) row + col * cols
   (d) col + row * rows

2. Structure padding digunakan untuk:
   (a) Mengurangi memory usage
   (b) Alignment optimization
   (c) Error detection
   (d) Security

3. Pointer arithmetic pada int pointer menambah:

(a) 1 byte

(b) 2 bytes

(c) 4 bytes

(d) 8 bytes

B. Essay

1. Jelaskan implementasi complete addressing modes untuk bahasa dengan arrays, structures, dan pointers!
2. Desain memory layout system yang efisien untuk dynamic data structures!

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat mengimplementasikan addressing modes untuk variabel dan arrays

☐ Saya dapat menghitung memory layout untuk structures dan unions

☐ Saya dapat melakukan pointer arithmetic operations

☐ Saya dapat mengimplementasikan array descriptor systems

☐ Saya memahami virtual memory address translation

☐ Saya dapat mendesain efficient memory layouts

Rangkuman

Bab ini membahas memory layout dan addressing modes, termasuk storage classes, addressing modes, array implementation, structure layout, pointer arithmetic, dan address translation. Mahasiswa belajar mengimplementasikan efficient memory management.

Poin Kunci:

- Memory layout menentukan bagaimana data disimpan dan diakses
- Addressing modes menyediakan berbagai cara mengakses data
- Array addressing memerlukan perhitungan offset yang tepat
- Structure layout mempertimbangkan alignment dan padding

- Pointer arithmetic memungkinkan efficient data access
- Virtual memory translation memisahkan logical dan physical addresses

Kata Kunci: ***Memory Layout***, ***Addressing Modes***, ***Array Addressing***, ***Structure Layout***, ***Pointer Arithmetic***, ***Virtual Memory***, ***Memory Alignment***

Daftar Pustaka

[1] Johns Hopkins University. *EN.601.428/628: Compilers and Interpreters*. Course syllabus and materials, taught by David Hovemeyer. 2024. URL: https://jhucompilers.github.io/fall2025/syllabus.html.