PP/018/SO/IV/R1

PETUNJUK PRAKTIKUM

SISTEM OPERASI

TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS AHMAD DAHLAN

2016

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM I

MODEL PEMROGRAMAN 1

A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Memahami komponen arsitektur komputer tingkat bawah.
2. Menggunakan simulator untuk membuat dan mengeksekusi instruksi dasar CPU.
3. Membuat instruksi untuk memindahkan data ke register, membandingkan isi register, memasukkan data ke stack, mengambil data dari stack, melompat ke lokasi alamat, menambahkan nilai dalam register.
4. Menjelaskan fungsi-fungsi dari register khusus CPU antara lain register PC, SR, dan SP.

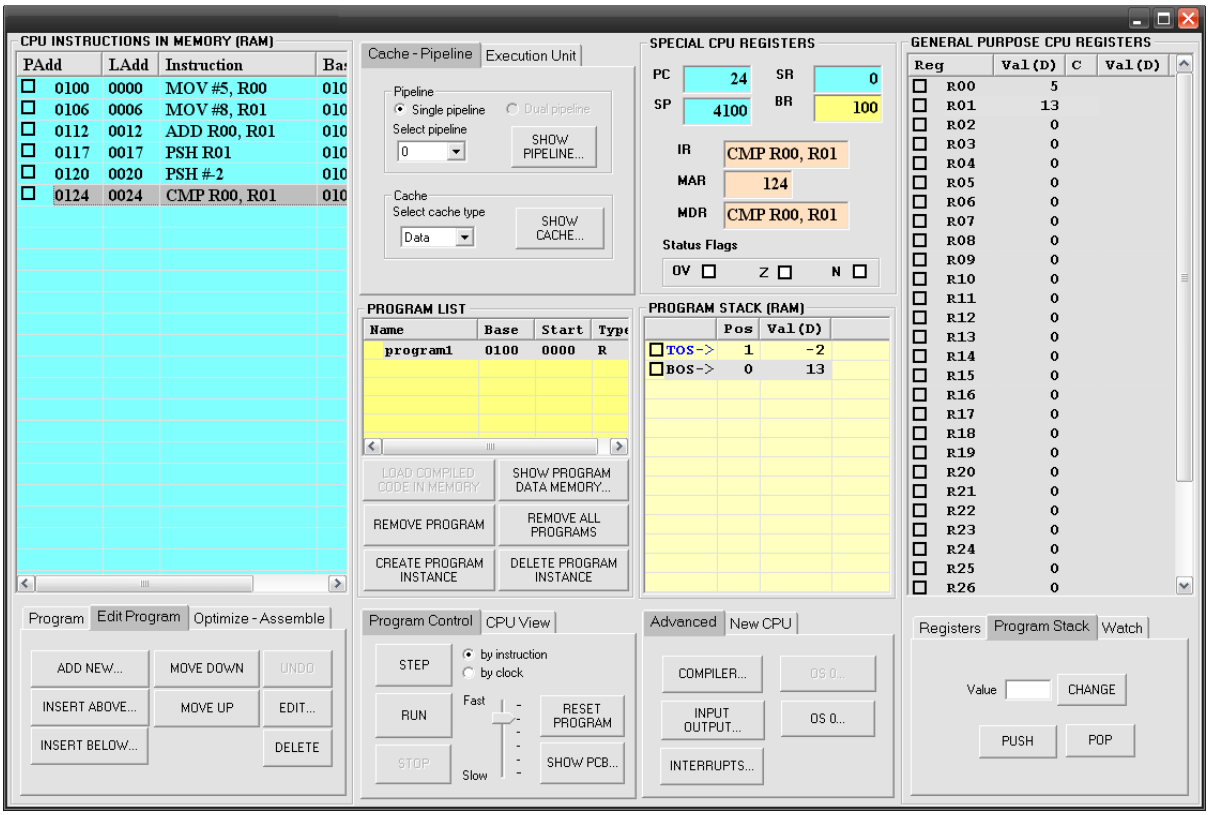
B. Dasar Teori

Model pemrograman arsitektur komputer mendefinisikan komponen-komponen arsitektur pada tingkat bawah (low level architectural components) yang mencakup:

- Set instruksi prosesor
- Register-register
- Jenis-jenis pengalamatan instruksi dan data
- *Interrupts* dan *exceptions*

Pada model pemrograman ini terdapat interaksi antar komponen diatas. Ini merupakan model pemrograman tingkat rendah (low-level) yang memungkinkan suatu komputasi terprogram.

C. Perangkat Simulator

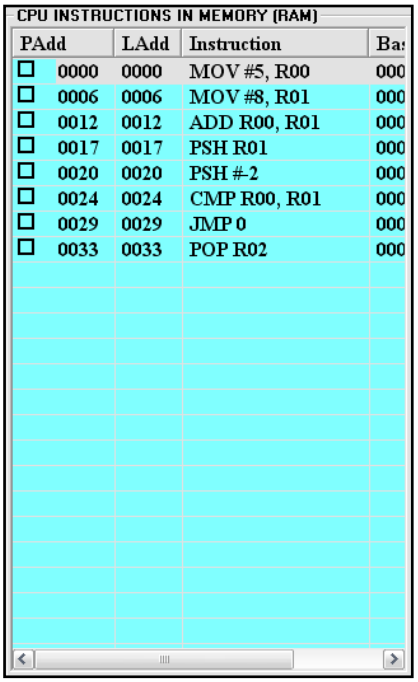


Gambar 1. Jendela utama simulator

Jendela utama terdiri dari beberapa tampilan, yang merepresentasikan beberapa bagian fungsional prosesor yang disimulasikan, yaitu:

- Instruction memory (RAM)
- Special registers
- Register set
- Program stack

Tampilan *instruction memory*

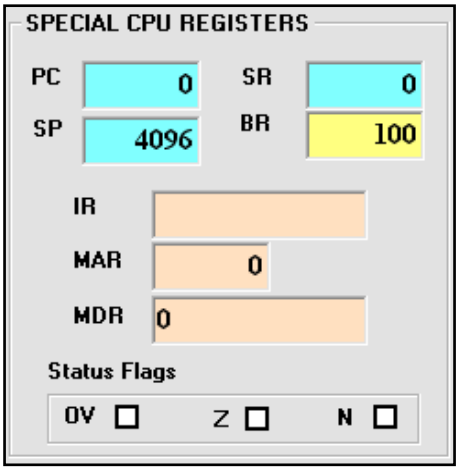


Tampilan ini berisi instruksi-instruksi dari program. Instruksi ditampilkan dalam urutan instruksi mnemonik low-level (format assembler), tidak dalam bentuk kode biner. Hal ini bertujuan untuk memperjelas dan membuat kode lebih mudah dibaca.

Tiap instruksi mempunyai dua alamat: alamat fisik (Padd) dan alamat logika (Ladd). Tampilan ini juga menyajikan alamat dasar (Base) terhadap tiap instruksi. Urutan instruksi pada program yang sama akan mempunyai alamat dasar yang sama.

Gambar 2. Tampilan *instruction view*

Tampilan *Special Registers*



Gambar 3. Tampilan Special Register

Tampilan ini menyajikan register-register dengan fungsi-fungsi khusus yang telah ditentukan:

PC : Program Counter, berisi alamat instruksi berikutnya yang akan dieksekusi.

IR : Instruction Register, berisi instruksi yang sedang dieksekusi saat ini.

SR : Status Register, berisi informasi yang memberikan hasil dari instruksi yang dieksekusi terakhir.

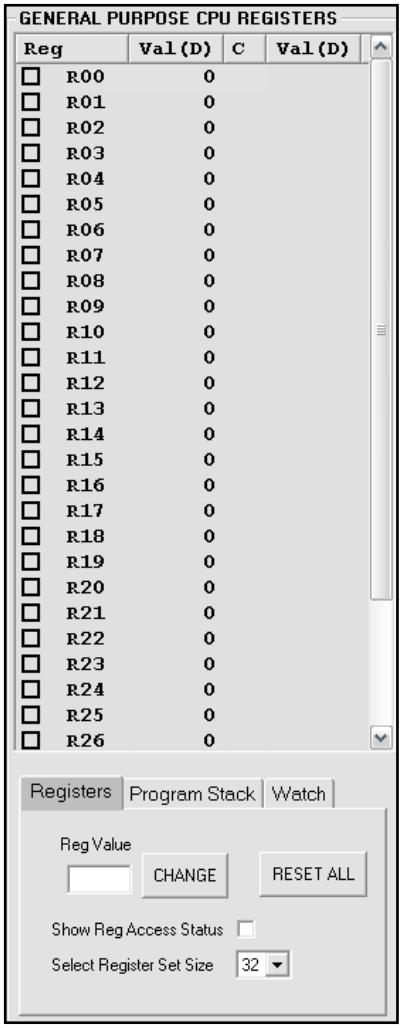
SP : Stack Pointer, register menunjuk ke nilai yang berada pada bagian atas stack.

BR : Base Register, berisi alamat dasar saat ini.

MAR : Memory Address Register, berisi alamat memori yang sedang diakses.

Status Bits : OV: Overflow; **Z**: Zero; **N**: Negative

Tampilan *Register Set*



Tampilan register set menunjukkan isi dari semua register-register tujuan umum (general-purpose), yang digunakan untuk menampung nilai sementara ketika instruksi program dieksekusi.

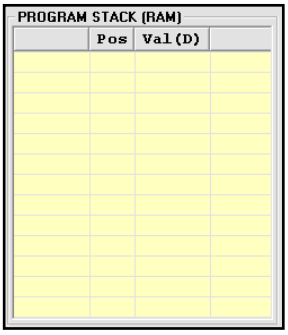
Arsitektur ini mendukung 8 hingga 64 register. Register-register ini sering digunakan untuk menyimpan nilai variabel program pada bahasa tingkat tinggi.

Tidak semua arsitektur memiliki register sebanyak ini. Beberapa lebih banyak (misalnya 128 register) dan beberapa lainnya lebih sedikit (misalnya 8 register). Pada semua kasus, register-register ini bekerja untuk tujuan yang sama.

Bagian ini menampilkan nama tiap register (Reg), nilai (Val), dan beberapa informasi lainnya untuk keperluan debug program. Kita juga dapat melakukan reset (RESET) atau mengisikan nilai register (CHANGE) secara manual.

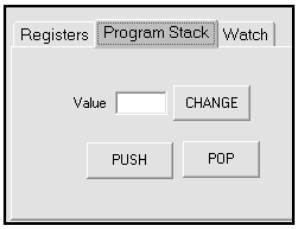
Gambar 4. Tampilan Register Set

Tampilan *Program Stack*



Program Stack menampung nilai sementara ketika instruksi dieksekusi. Stack menggunakan struktur data LIFO (last-In-First-Out). Stack sering digunakan untuk efisiensi *interrupt handling* dan *sub-routine call*.

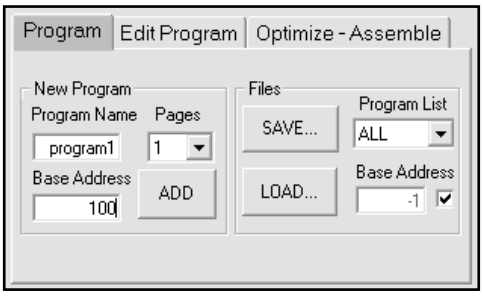
Gambar 5. Tampilan Program Stack



Instruksi PUSH dan POP digunakan untuk menyimpan dan mengambil nilai pada bagian teratas stack.

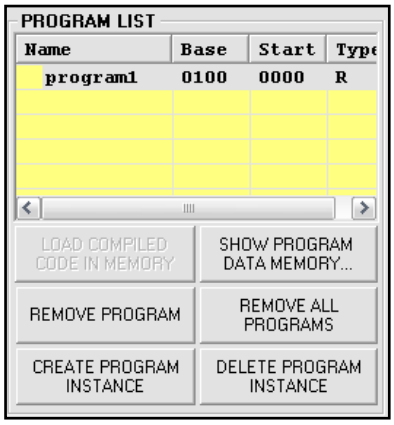
D. Persiapan Praktikum

Pertama-tama kita perlu menempatkan sejumlah instruksi pada tampilan instruction memory (menggambarkan RAM pada mesin yang nyata) sebelum melakukan eksekusi instruksi. Berikut adalah cara menempatkan instruksi ke instruction memory :



Pada tampilan program instruction, masukkan nama program (misal: program1), dan juga alamat dasar (untuk praktikum ini beri alamat dasar nilai 100). Klik tombol ADD. Maka program baru dengan namanya akan dimasukkan ke tampilan Program List seperti yang disajikan gambar 7. Gunakan tombol SAVE.../ LOAD... untuk menyimpan dan mengambil instruksi dari file.

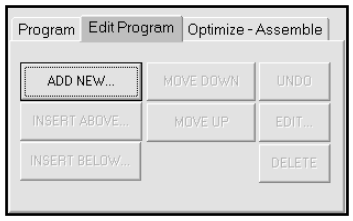
Gambar 6. Tampilan Program Instructions.



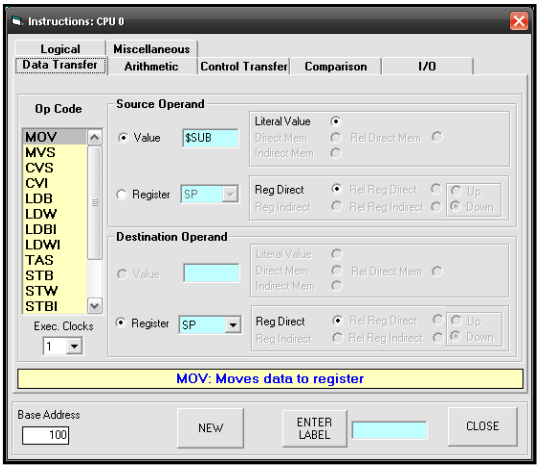
Gunakan tombol REMOVE PROGRAM untuk menyingkirkan program yang disorot. Gunakan tombol REMOVE ALL PROGRAMS untuk menyingkirkan semua program yang ada di daftar. Sebagai catatan, ketika program disingkirkan maka instruksi-instruksi yang terkait dengannya juga akan dihapus dari tampilan instruction memory.

Gambar 7. Tampilan Program List

E. Langkah-langkah Praktikum



Untuk memasukkan instruksi ke CPU, klik tombol ADD NEW... yang akan menampilkan jendela Instructions: CPU0.



Gunakan jendela ini untuk memasukkan instruksi. Pada **Appendix** tersedia beberapa instruksi yang dikenali oleh simulator ini dan terdapat pula contoh penggunaannya.

a. Transfer data

1. Buatlah instruksi yang memindahkan (move) angka 5 ke register R00.
2. Eksekusi instruksi diatas (dengan klik dua kali pada tampilan instruction memory).
3. Buatlah instruksi yang memindahkan angka 8 ke register R01.
4. Eksekusilah.
5. Amati isi R00 dan R01 pada tampilan Register Set.

b. Aritmatika

6. Buatlah suatu instruksi yang menambahkan (add) isi R00 dan R01.
7. Eksekusilah.
8. Amati dimana hasil penjumlahan tersebut disimpan.

c. Stack Pointer (SP)

9. Buatlah instruksi yang menaruh hasil diatas pada program stack, kemudian eksekusilah.
10. Buatlah instruksi untuk menaruh (push) angka -2 pada stack teratas dan eksekusilah.
11. Amati nilai register SP.

d. Pembanding

12. Buatlah instruksi untuk membandingkan nilai register R00 dan R01.
13. Eksekusilah.
14. Amati nilai register SR.
15. Amati bit status OV/Z/N pada status register.

e. Program Counter (PC)

16. Buatlah instruksi untuk melakukan unconditionally jump ke instruksi yang pertama.
17. Eksekusilah.
18. Amati nilai register PC. Apa yang ditunjuk olehnya?

f. Alamat fisik dan alamat logika

19. Amati nilai pada kolom Padd dan Ladd. Apa arti nilai-nilai yang ditunjukkan oleh keduanya? Apa perbedaannya?
20. Apa perbedaan antara nilai Ladd pada instruksi yang pertama dengan nilai LAdd pada instruksi yang kedua? Apa arti nilai tersebut?

g. Stack Pointer (SP)

21. Buatlah instruksi untuk mengambil (pop) nilai teratas dari program stack ke register R02.
22. Eksekusilah.
23. Amati nilai pada register SP.
24. Buatlah suatu instruksi untuk mengambil nilai teratas dari program stack ke register R03.
25. Eksekusilah.
26. Amati nilai pada register SP.
27. Eksekusi lagi instruksi yang terakhir. Apa yang terjadi? Jelaskan.

h. Status Register (SR)

28. Buatlah suatu instruksi pembanding (compare) yang membandingkan nilai dalam register R04 dan R05.
29. Sisipkan secara manual dua nilai yang sama pada register R04 dan R05.
30. Eksekusi instruksi pembanding pada langkah 28 diatas.
31. Manakah diantaras status flag OV/Z/N yang dalam kondisi set? Mengapa?
32. Sisipkan secara manual nilai dalam register R05 yang lebih besar dari register R04.
33. Eksekusilah instruksi pembanding pada langkah 28 diatas.
34. Manakah diantaras status flag OV/Z/N yang dalam kondisi set? Mengapa?
35. Sisipkan secara manual nilai dalam register R04 yang lebih besar dari register R05.
36. Eksekusilah instruksi pembanding pada langkah 28 diatas.
37. Manakah diantaras status flag OV/Z/N yang dalam kondisi set? Mengapa?

i. Jump if...

38. Buatlah instruksi yang akan melompat (jump) ke instruksi yang pertama jika nilai dalam register R04 dan R05 sama.
39. Ujilah instruksi ini dengan mengisikan nilai yang sesuai pada register R04 dan register R05 kemudian eksekusilah instruksi pembanding, lanjutkan dengan mengeksekusi instruksi lompat.
40. Simpan instruksi dalam Instruction Memory ke file dengan klik pada tombol SAVE.

----- end -----

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM II

MODEL PEMROGRAMAN 2

A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Menggunakan simulator CPU untuk membuat instruksi-instruksi dasar CPU.
2. Menggunakan simulator untuk mengeksekusi instruksi-instruksi dasar CPU.
3. Menggunakan mode pengalamatan langsung dan tidak langsung.
4. Membuat iterative loop.
5. Mengenal kode ASCII.
6. Membuat sub-routine, sub-routine call, dan return dari sub-routine.

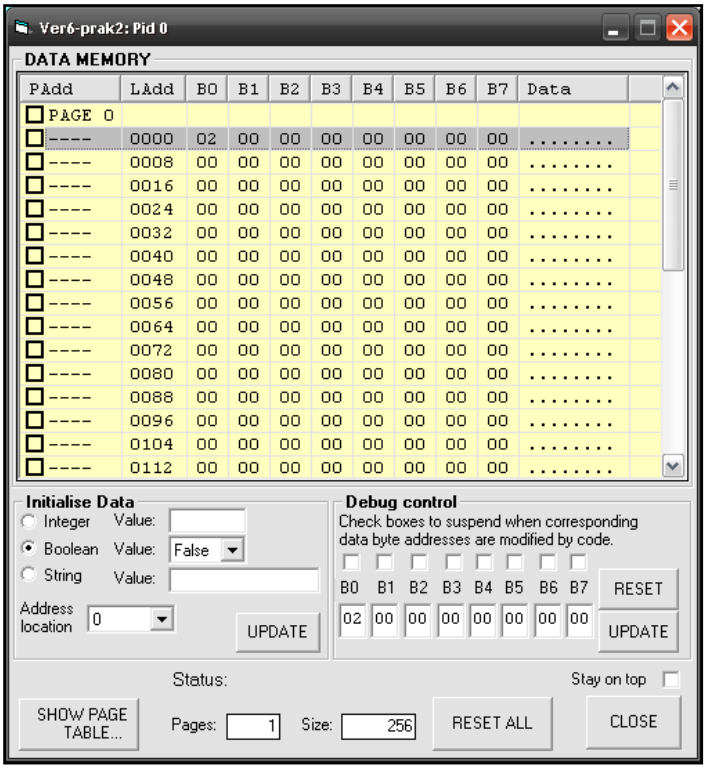
B. Dasar Teori

Model pemrograman arsitektur komputer mendefinisikan komponen-komponen arsitektur pada tingkat bawah (low level architectural components) yang mencakup:

- Set instruksi prosesor
- Register-register
- Jenis-jenis pengalamatan instruksi dan data
- *Interrupts* dan *exceptions*

Pada model pemrograman ini terdapat interaksi antar komponen diatas. Ini merupakan model pemrograman tingkat rendah (low-level) yang memungkinkan suatu komputasi terprogram.

C. Persiapan Praktikum



Gambar 1. Tampilan *Data Memory*

Pada praktikum kali ini anda perlu melihat isi program yang dikirimkan ke memori.

Untuk melihatnya klik tombol SHOW PROG DATA MEMORY... pada tampilan PROGRAM LIST. Isi memori akan ditampilkan dalam suatu jendela yang terpisah seperti yang tampak pada gambar 1.

Untuk alasan memudahkan, alamat disajikan dalam format desimal dan data memori ditampilkan dalam heksadesimal.

D. Langkah-langkah Praktikum

Untuk memasukkan instruksi ke CPU, klik tombol ADD NEW... yang akan menampilkan jendela Instructions: CPU0.

1. Data Transfer

a. Pada Appendix, temukan instruksi yang digunakan untuk menyimpan (store) suatu byte of data ke suatu alamat memori.
b. Gunakan instruksi tersebut untuk menyimpan 1 byte angka 65 ke lokasi alamat 20 (semua angka ini adalah desimal). Ini adalah contoh pengalamatan langsung.
c. Buatlah instruksi untuk memindahkan angka 22 ke register R01 dan eksekusilah.
d. Buatlah suatu instruksi untuk menyimpan angka 51 ke lokasi alamat yang tersimpan pada register R01 dan eksekusilah. Ini adalah contoh pengalamatan tidak langsung.
e. Pastikan bahwa byte yang dituliskan berada pada lokasi alamat yang benar. Pada tahap ini mestinya akan tampak (di jendela DATA MEMORY) huruf A dan angka 3 di kolom Data.
f. Jelaskan yang dimaksud dengan pengalamatan secara langsung dengan tidak langsung.

2. Loop

g. Sekarang kita membuat loop: pertama-tama set R2 berisi 0 (nol). Naikkan nilai R02 sebesar 1. Lakukan perbandingan (*comparison*), jika nilai R02 adalah 5 maka keluar dari loop ini dan hentikan program; selain itu lanjutkan loop. Pastikan loop ini bekerja.

3. Memori

h. Kemudian masukkan suatu teks pendek ke memori. Klik dan sorot lokasi memori 0024 (Pada kolom PAdd). Isikan 03, ‘h, ‘e, ‘l, ‘l, ‘o, 0D, 0A ke dalam kotak B0 sampai B7, dan klik tombol UPDATE. Seharusnya dalam memori akan tampil teks “hello” (yang dimulai dari lokasi alamat 24). Apa arti dua byte heksadesimal 0D0A (akan terjawab pada langkah j)?
i. Modifikasikan loop diatas sehingga teks hello ditampilkan 5 kali pada konsol. Untuk melakukan ini sisipkan instruksi OUT (lihat Appendiks, atur pengalamatan secara langsung (*direct*)) pada lokasi yang sesuai dalam loop.
j. Pastikan ketika loop dieksekusi maka akan tampil teks “hello” sebanyak 5 kali dengan posisi dibawah teks sebelumnya. Untuk melihat ini, klik pada tombol INPUT/OUTPUT... pada bagian ADVANCED.

Gambar 2. Tampilan output.

Sekarang, beralih ke jendela memori dan gantilah 0D dan 0A dalam kotak B6 dan B7 dengan nilai 00 (gunakan tombol UPDATE untk mengganti nilai tersebut) dan ulangi menampilkan loop. Mengapa hasil tampilannya berbeda?

4. Sub Routine

k. Ubahlah loop menjadi suatu sub-routine. Kemudian tambahkan instruksi-instruksi yang diperlukan untuk memanggil (call) sub-routine tersebut (lihat Appendiks).

l. Amati isi register PC dan Program Stack tepat sebelum sub-routine call. Amati lagi ketika tepat setelah *sub-routine return instruction* dieksekusi. Jelaskan hasil pengamatan Anda.

m. Simpan instruksi-instruksi yang ditampilkan pada Instruction Memory ke file dengan menekan tombol SAVE....

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM III

MODEL PEMROGRAMAN 3

A. Tujuan

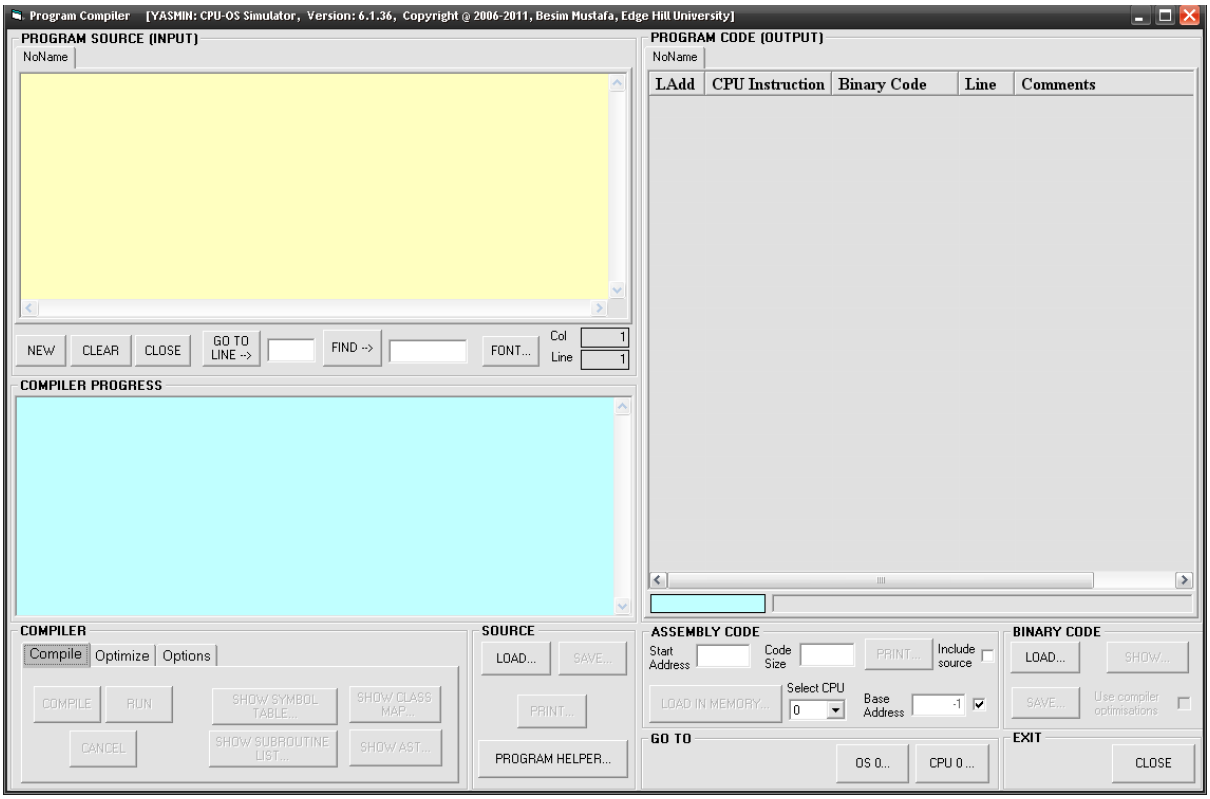
Pada akhir praktikum ini, peserta dapat:

1. Menjelaskan bagaimana variabel umum suatu program di simpan.
2. Membedakan berbagai tipe statement program tingkat tinggi.
3. Memahami kode tingkat rendah terkait dengan statement program.
4. Menjelaskan bagaimana sub-routine program bekerja.
5. Menggunakan simulator untuk membuat user interrupt

B. Dasar Teori

Program bahasa tingkat tinggi (High-level language) terdiri dari variabel-variabel yang menyimpan nilai data dan banyak statement program sebagai suatu algoritma. Statement-statement ini seringkali mengendalikan aliran eksekusi program dalam kondisi tertentu. Panggilan terhadap suatu sub-routine dan interrupt, mengubah aliran sekuensial suatu eksekusi program.

C. Perangkat Simulator



Gambar 1. Jendela utama *compiler*

Pada jendela compiler terdapat 3 sub jendela utama:

- Program Source – semua high-level source statement ditampilkan disini.
- Compiler Progress – informasi pada progres kompilasi tampil disini.
- Program Code – kode assembly yang dihasilkan oleh compiler tampil disini.

D. Langkah-langkah Praktikum

1. Masukkan kode sumber berikut ini, kemudian compile kode tersebut.

```
program Test1
   var IntVar integer
   var BoolVar boolean
   var StrVar1 string (5)
   var StrVar2 string(20)

   IntVar = 6
   BoolVar = true
   StrVar1 = "Hello"
   StrVar2 = "And again"
end
```

   Klik pada tombol SYMBOL TABLE.... akan tampil jendela SYMBOL TABLE. Amati informasi-informasi yang disajikan. Buat catatan pada field TYPE, SIZE, dan ADDRESS untuk masing-masing entry dalam tabel tersebut.
   Berikutnya, LOAD IN MEMORY program yang telah di compile tadi. Pada jendela CPU simulator klik tombol SHOW PROG DATA MEMORY.... Isi data dari program akan ditampilkan. Klik STAY ON TOP. Kemudian jalankan (run) program pada kecepatan maksimum. Amati isi dari memori, perhatikan pada lokasi alamat yang telah anda catat tadi.

2. Masukkan kode statement berikut ini

```
Program Test2
   n = 0
   i = n + 1
   p = i * 3
   writeln (“ n=”, n, ” i=”, i, ” p=”, p)
end
```

   Compile program diatas. Sekarang amati kode yang dihasilkan pada jendela PROGRAM CODE. Anda tidak perlu menganalisa secara detil. Hitung jumlah instruksi lompat (jump, kode yang diawali dengan huruf J) dan catatlah. Jelaskan jenis statement program yang digunakan program ini.

3. Masukkan statement sumber berikut ini.

```
Program Test3
   n = 0
   if n < 5 then
   p = p + 1
   end if
end
```

Compile program diatas. Amati kode yang dihasilkan. Berapa banyak instruksi lompat yang ada? Jelaskan pendapat anda tujuan instruksi lompat (jump) pada kode ini? Jenis statement apakah “if”?

4. Masukkan statement sumber berikut ini.

```
program Test4
   p = 1
   for n = 1 to 10
   p = p * 2
   next
end
```

Compile program diatas. Sekarang amati kode yang dihasilkan. Berapa banyak instruksi lompat (jump) yang ada? Jelaskan pendapat anda tujuan instruksi lompat (jump) pada kode ini? Jenis statement apakah “for”? Dapatkah anda memberi contoh statement lain untuk jenis ini?

5. Masukkan statement sumber berikut ini.

```
Program Test5
   sub One
         writeln(“I am sub One”)
   end sub

   sub Two
         call One
         writeln(“I am sub Two”)
   end sub

   call Two
End
```

Compile program diatas. Kemudian, LOAD IN MEMORY. Pada jendela CPU Simulator lakukan langkah-langkah berikut :
Klik pada frame PROGRAM LIST tombol RESET. Sekarang anda akan mengeksekusi program ini secara manual per instruksi. Untuk melakukannya double-klik instruksi yang sedang disorot. Berarti anda akan memulai dari instruksi MSF, kemudian CAL, dst. Sembari anda mengeksekusi program dengan cara tersebut, lakukan pengamatan sebagai berikut:
Buat catatan terhadap isi frame PROGRAM STACK setelah suatu instruksi CAL atau instruksi RET dieksekusi. Terus menjalankan eksekusi instruksi hingga mencapai instruksi HLT.
Jelaskan apa yang terjadi setiap suatu instruksi CAL atau RET dieksekusi dan bagaimana pengaruhnya terhadap isi PROGRAM STACK.

6. Masukkan statement sumber berikut ini.

```
program Test6
   sub Any
      n = 0
   end sub

   sub MeToo intr 5
        writeln("Me too, me too!")
   end sub

   do
   loop
end
```

Compile program diatas. Perhatikan pada kode yang dihasilkan. Dimana alamat sub-routine "MeToo" dimulai? Catatlah angka tersebut.
Kemudian, LOAD IN MEMORY. Pada jendela CPU Simulator klik pada tombol INTERRUPTS.... maka akan tampil jendela INTERRUPTS. Catatlah angka beserta kotak yang menampilkannya. Jelaskan arti angka ini.
Klik STAY ON TOP pada jendela INTERRUPTS. Klik tombol INPUT/OUTPUT... dan atur pula agar jendela console STAY ON TOP.
Kemudian atur kecepatan CPU Simulator hingga maksimum, dan jalankan (RUN) program (atau bisa juga menggunakan STEP). Buatlah catatan terhadap hasil pengamatan. Jelaskan tujuan utama loop statement "do" pada program ini.
Berikutnya, klik tombol TRIGGER pada jendela INTERRUPTS sambil mengamati jendela Console. Catatlah hasil pengamatan anda.
Kurangi laju CPU Simulation (kira-kira diatas setengah bar). Picu (trigger) interrupt dan amati isi PROGRAM STACK. Anda dapat klik tombol STOP ketika melihat sejumlah kode tampil pada stack ini sehingga anda dapat mengamatinya dengan leluasa. Jelaskan hasil observasi anda!

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM IV

Thread

A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Menulis kode sumber yang membuat thread
2. Menampilkan daftar proses/ thread dan pohon proses yang menggambarkan hubungan antar proses induk/anak.
3. Memodifikasi kode sumber untuk membuat versi tanpa thread dan membandingkannya dengan versi yang menggunakan thread
4. Memahami alasan penggunaan thread.

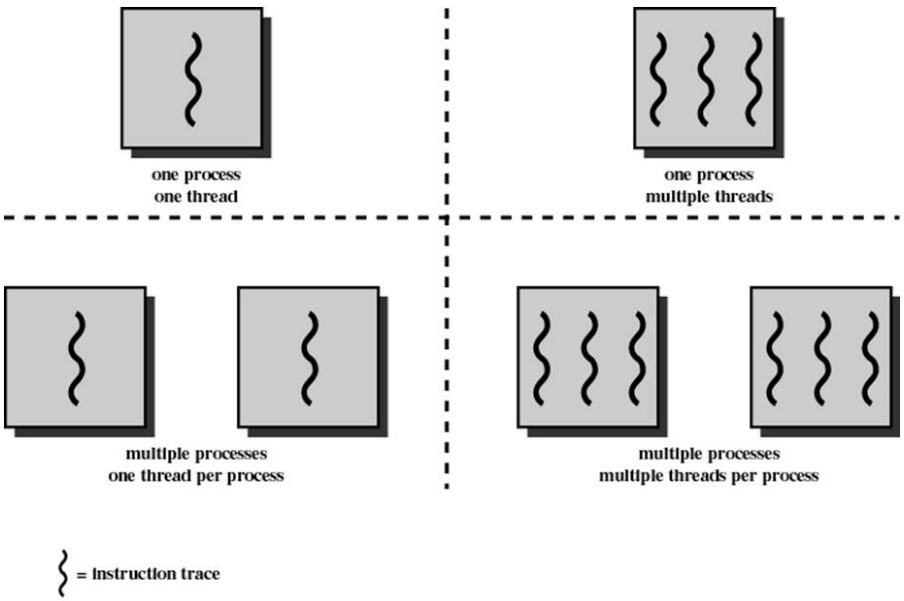
B. Dasar Teori

Konsep proses mengandung dua karakteristik:

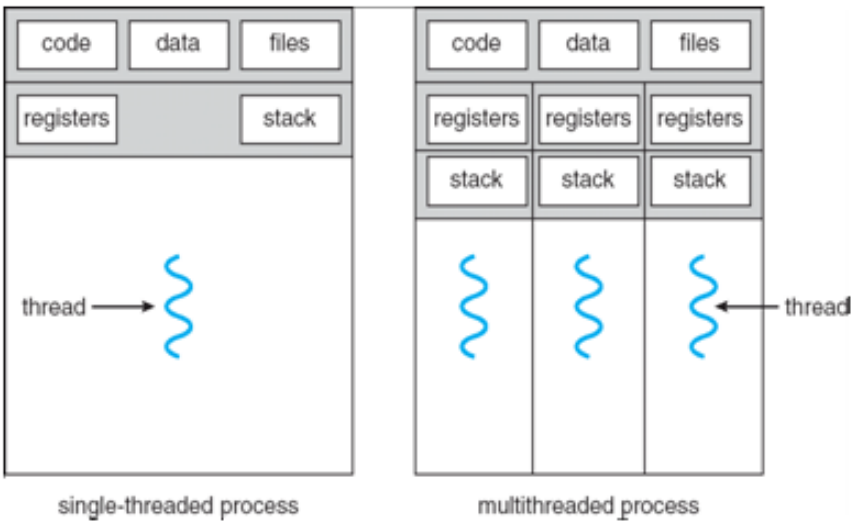
1. Unit kepemilikan sumber daya
2. Unit pengiriman (dispatching)

Untuk membedakan kedua karakteristik itu, unit dispatching biasanya disebut sebagai **thread**, atau *lightweight process*, sedangkan unit kepemilikan sumber daya biasanya masih disebut sebagai proses atau task (Stallings, 2003).

Dengan begitu pada tiap *thread* terdapat ID thread, program counter, register, dan stack (terkait unit dispatching). Namun saling berbagi dengan thread yang lain (dalam satu proses) terhadap kepemilikan sumber daya yang sama.



Gambar 1. Thread dan Proses



Gambar 2. Thread dan sumber daya yang dimiliki

C. Langkah-langkah Praktikum

1. Jalankan Simulator. Tuliskan kode berikut ini ke compiler.

```
program ThreadTest

      sub thread1 as thread
            writeln("In thread1")
            while true
            wend
      end sub

      sub thread2 as thread
            call thread1
            writeln("In thread2")
            while true
            wend
      end sub

      call thread2
      writeln("In main")
      do
      loop
end
```

Coba anda perkirakan hasilnya apabila program tersebut dijalankan.

2. Compile kode diatas. Muatkan ke memori.
3. Melalui CPU Simulator, tampilkan jendela konsol dengan klik tombol INPUT/OUTPUT... aktifkan STAY ON TOP.
4. Beralihlah ke jendela OS Simulator, buat satu proses dengan CREATE NEW PROCESS.
5. Pastikan jenis penjadwalan adalah ROUND ROBIN, 10 ticks dengan kecepatan simulasi maksimum.

6. Tekan tombol START dan pada waktu yang bersamaan **amati tampilan pada jendela konsol**. Jelaskan yang terjadi, apakah seperti yang anda perkirakan?

7. Kembali ke jendela OS Simulator, berapa jumlah proses/ thread yang tampak? Dapatkah anda bedakan antara proses dengan thread? Jelaskan.

8. Klik tombol VIEW PROCESS LIST... Amati dan beri penjelasan hubungan antara induk/anak.

9. Modifikasi kode sumber diatas dengan menghapus kode “as thread”. Compile kode dan muatkan ke memori. Buat proses, dan jalankan OS simulator.

10. **Amati** tampilan pada layar konsol. Jelaskan perbedaan hasil yang ditampilkan sekarang, dibandingkan bila program menggunakan “as thread”.

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM IV

Penjadwalan Proses I

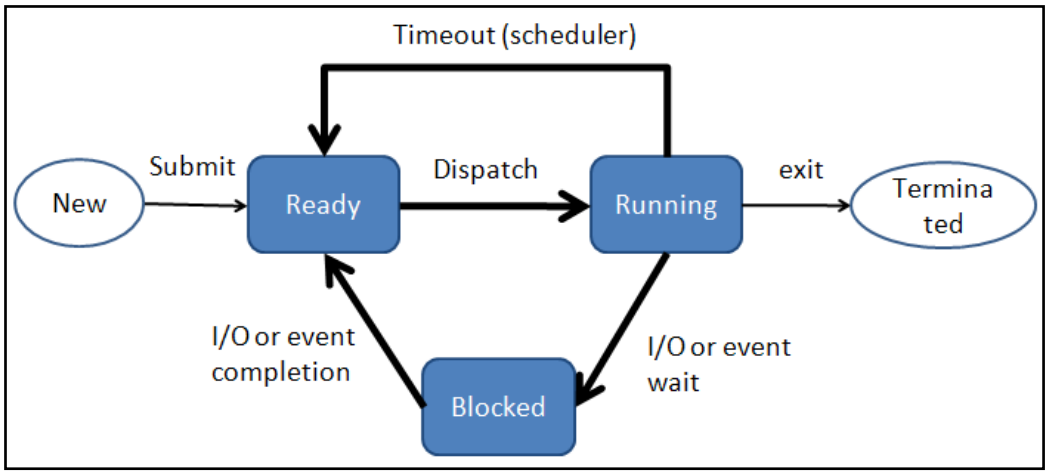
A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Menggunakan simulator sistem operasi
2. Memahami konsep transisi keadaan proses (process state)
3. Memahami penggunaan berbagai jenis penjadwalan proses.
4. Memahami fungsi dan manfaat *system log* pada sistem operasi

B. Dasar Teori

Proses melewati serangkaian keadaan diskrit. Beragam kejadian dapat menyebabkan perubahan/ berpindahnya keadaan proses. Gambar 1 menunjukkan diagram tiga keadaan dasar transisi keadaan proses.



Gambar 1. Diagram tiga keadaan Proses

Penjadwalan proses merupakan kumpulan kebijakan dan mekanisme di sistem operasi yang berkaitan dengan urutan kerja yang dilakukan sistem komputer. Penjadwal (*scheduler*) bertugas memutuskan hal-hal berikut:

- Proses yang harus berjalan
- Kapan dan selama berapa lama proses berjalan

Sasaran utama penjadwalan proses adalah:

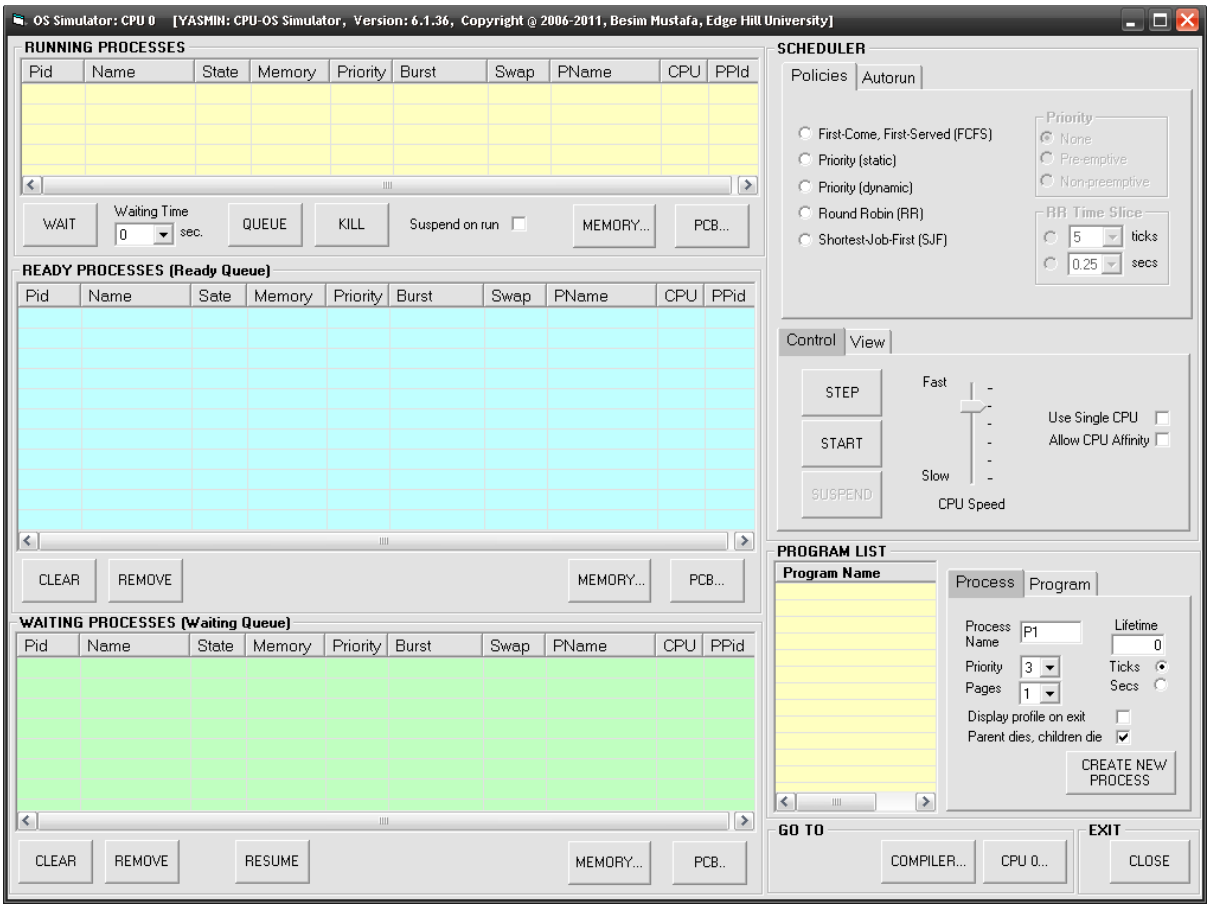
- Pada semua sistem
  - **Fairness** – mengatur proses dalam penggunaan bersama sumber daya CPU secara adil
  - **Policy enforcement** – memastikan kebijakan yang diterapkan berjalan
  - **Balance** – menjaga semua bagian sistem dalam keadaan sibuk
- Pada sistem batch
  - **Throughput** – memaksimalkan jumlah job per jam (jobs per hour)
  - **Turnaround time** – meminimalkan waktu dari mulai (submission) hingga berakhir (termination)
  - **CPU utilization** – mengupayakan CPU selalu sibuk sepanjang waktu

- Pada sistem interaktif
    - **Response time** – merespon permintaan (request) dengan cepat
    - **Proportionality** – memenuhi harapan pengguna (user's expectation)
- Pada sistem real-time
    - **Meeting deadlines** – menghindari kehilangan data
    - **Predictablility** – menghindari penurunana kualitas dalam sistem multimedia

Klasifikasi algoritma penjadwalan, berdasarkan dapat/ tidaknya suatu proses diambil alih secara paksa:

- Algoritma yg menerapkan Non preemptive (tidak dapat diambil alih secara paksa), antara lain:
    - FIFO (first in first out) atau disebut juga FCFS (First Come First Serve),
    - SJF (shortest job first),
    - HRN (highest ratio next),
- Algoritma yg menerapkan Preemptive (dapat diambil alih secara paksa), antara lain:
    - RR (round robin),
    - MFQ (multiple feed back queues)
    - SRF (shortest remaining first),
    - PS (priority schedulling)
    - GS (Guaranteed Schedulling)

C. Perangkat Simulator



Gambar 2. Jendela utama *OS Simulator*

Pada jendela OS Simulator terdapat 6 sub jendela utama:

- Ready Processes
- Waiting Processes
- Running Processes
- (Scheduller) Policies
- (Scheduller) Control
- Program List

D. Langkah-langkah Praktikum

Buatlah kode sumber berikut menggunakan CPU Simulator. Simpan dengan nama FORNEXTLOOP.sas.

CPU INSTRUCTIONS IN MEMORY (RAM)

PAdd	LAdd	Instruction	Ba
0000	0000	MOV #0, R01	000
0006	0006	MOV #1, R02	000
0012	0012	MOV #40, R04	000
0018	0018	CMP R04, R02	000
0023	0023	JGT 53	000
0027	0027	MOV R01, R03	000
0032	0032	ADD #1, R03	000
0038	0038	MOV R03, R01	000
0043	0043	ADD #1, R02	000
0049	0049	JMP 18	000
0053	0053	HLT	000

Gambar 3. Kode untuk FORNEXTLOOP.sas

1. Membuat dan Menjalankan Proses

- Pada jendela OS Simulator pastikan program FORNEXTLOOP telah dimuatkan pada sub jendela PROGRAM LIST. Bila belum, lakukan pemuatan (load).
- Klik tombol CREATE PROCESS. Jelaskan akibat/ hasil penekanan tombol tersebut. Sekali suatu proses dibuat, berarti proses ini menjadi kandidat untuk dijadwalkan berjalan oleh sistem operasi. Pada sistem operasi yang nyata, penjadwal (scheduler) selalu aktif mencari proses berikutnya, bila tersedia, akan dijalankan sesegera mungkin begitu CPU siap.
- Pada OS simulator pengguna harus menekan tombol start secara manual untuk mengaktifkan penjadwalan. Dengan begitu, klik tombol START, pada sub jendela CONTROL.
- Jelaskan apa yang terjadi! Dan informasi apa yang ditampilkan terkait suatu proses?
- Anda dapat mempercepat eksekusi proses dengan menggeser pengatur ke label FAST. Setelah beberapa saat maka proses akan berakhir dengan sendirinya.

2. Operating System Log

- Simulator memiliki log peristiwa-peristiwa penting. Untuk melihatnya, klik tombol VIEW LOG... pada sub jendela CONTROL.
- Jelaskan infomasi apa saja yang ditampilkan pada jendela log? Sebagai catatan, tampilan log ini spesifik untuk simulator ini, sedangkan pada sistem operasi yang nyata bisa jadi menyajikan informasi yang berbeda ataupun sama.

3. Percobaan pada beberapa algoritma penjadwalan

- Jelaskan pengertian algoritma!

a. FCFS

1. Pilih jenis penjadwalan First Come, First Serve (FCFS)
2. Buatlah dua proses
3. Atur kecepatan eksekusi pada FAST
4. Jalankan penjadwalan (Start).
5. Tunggu hingga semua proses berakhir dengan normal.
   Jelaskan perilaku jenis penjadwalan ini (FCFS).

b. PRIORITY

6. Pilih jenis penjadwalan PRIORITY (non-preemptive).
7. Buatlah proses dengan prioritas 3.
8. Buatlah proses dengan prioritas 2
9. Buatlah proses dengan prioritas 4
10. Buatlah proses dengan prioritas 3
    Jelaskan perilaku jenis penjadwalan ini (Priority).

c. ROUND ROBIN

11. Atur kecepatan proses eksekusi pada FAST
12. Jalankan penjadwalan.
13. Tunggu hingga semua proses berakhir secara normal.
14. Pilih jenis penjadwalan ROUND ROBIN.
15. Pilih "10 ticks" sebagai slot waktu.
16. Buatlah tiga proses.
17. Jalankan penjadwalan.
18. Tunggu sebentar (sekitar 2 detik) kemudian klik tombol SUSPEND pada sub jendela CONTROL.
19. Pilih proses yang berada pada sub jendela READY PROCESSES, dan klik tombol PCB...
    Jelaskan apa yang anda lihat?
    Amati angka yang tersaji terhadap register PC. Apa pentingnya angka ini?
    Informasi penting apa lagi yang anda temukan proses ini yang tersaji melalui jendela yang ditampilkan?
20. Sekarang, klik tombol RESUME.
21. Tunggu hingga semua proses berhenti secara normal.
    Jelaskan perilaku jenis penjadwalan ini (round robin).

d. PRIORITY, PRE-EMPTIVE

22. Pilih jenis penjadwalan PRIORITY (pre-emptive).
23. Atur kecepatan eksekusi hingga sangat mendekati SLOW.
24. Buatlah proses dengan prioritas 2
25. Buatlah proses dengan prioritas 3
26. Jalankan penjadwalan.
27. Sekarang, buatlah proses dengan prioritas 1.
    Proses manakah yang sedang berjalan (running)? Proses manakah yang dalam antrian ready? Jelaskan perilaku jenis penjadwalan ini.

e. FCFS, dengan waiting time

28. Tunggu hingga semua proses (dari aktivitas d) berakhir secara normal.
29. Pilih jenis penjadwalan FCFS
30. Buatlah dua proses
31. Atur kecepatan pada FAST.
32. Jalankan Penjadwal.
33. Sekarang, pada sub jendela RUNNING PROCESS, atur WAITING TIME 10 detik, kemudian klik tombol WAIT.
34. Amati perilaku proses yang berjalan, bandingkan antara sebelum dan sesudah anda menekan tombol WAIT (anda harus menunggu sebentar untuk melihat hasilnya). Jelaskan perilaku proses ini. Bagaimana **transisi keadaan** yang ditempuh?

f. LOG

35. Tunggu hingga semua proses berakhir secara normal.
36. Sekarang, tampilkan log dan telusuri hasil dari aksi yang anda lakukan sejauh ini. Kemudian berikan analisa anda mengenai manfaat log tersebut.

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM VI

Sinkronisasi Proses

A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Mengkompilasi kode sumber dengan thread yang berbagi akses ke area global yang tidak diproteksi.
2. Menunjukkan bahwa menulis ke area global yang tidak diproteksi dapat menimbulkan efek samping yang tidak diinginkan ketika diakses oleh thread.
3. Mengkompilasi kode sumber dengan thread yang tersinkronisasi.
4. Menunjukkan bahwa hardware menyediakan bantuan untuk membuat critical region.
5. Mengkompilasi kode sumber dengan mutex untuk membuat wilayah exclusion.
6. Menunjukkan bahwa OS menyediakan bantuan untuk membuat critical region.

B. Dasar Teori

Proses konkuren mengakses sumber daya global pada saat yang bersamaan dapat menghasilkan efek samping yang tidak diinginkan. Perangkat keras atau sistem operasi komputer dapat menyediakan mutual exclusion ketika sumber daya tersebut digunakan bersama.

C. Langkah-langkah Praktikum

Jalankan Simulator.

1. Tuliskan kode berikut ini pada compiler.

```
program CriticalRegion
        var g integer

        sub thread1 as thread
             writeln("In thread1")
             g = 0
             for n = 1 to 20
                  g = g + 1
             next
             writeln("thread1 g = ", g)
             writeln("Exiting thread1")
        end sub

        sub thread2 as thread
             writeln("In thread2")
             g = 0
             for n = 1 to 12
                  g = g + 1
             next
             writeln("thread2 g = ", g)
             writeln("Exiting thread2")
        end sub

        writeln("In main")

        call thread1
        call thread2

        wait

        writeln("Exiting main")
end
```

Kode diatas akan membuat dua thread yaitu thread1 dan thread2. Masing-masing thread menaikkan nilai variabel global g dalam tiap loop. Ketika dua loop selesai nilai apa yang anda perkirakan dalam variabel g untuk dua kasus tersebut?

Dapatkah anda menduga fungsi statement **wait**?

a. Sekarang compile kode tersebut, muatkan dalam memori.
b. Tampilkan jendela konsol, aktifkan STAY ON TOP.
c. Beralihlah ke OS simulator, akan tampak CriticalRegion pada PROGRAM LIST.
d. Buatlah proses dengan ketentuan : ROUND ROBIN, 10 ticks, kecepatan maksimum.
e. Tekan START.
f. Ketika program berhenti, catatlah dua nilai g yang ditampilkan. Apakah nilai ini seperti yang anda perkirakan tadi? Bagaimana pendapat anda.

2. Sekarang kita memodifikasi program seperti dibawah ini. Bagian yang dimodifikasi ditandai dengan tebal (bold).

```
program CriticalRegion
        var g integer

        sub thread1 as thread synchronise
             writeln("In thread1")
             g = 0
             for n = 1 to 20
                  g = g + 1
             next
             writeln("thread1 g = ", g)
             writeln("Exiting thread1")
        end sub

        sub thread2 as thread synchronise
             writeln("In thread2")
             g = 0
             for n = 1 to 12
                  g = g + 1
             next
             writeln("thread2 g = ", g)
             writeln("Exiting thread2")
        end sub

        writeln("In main")

        call thread1
        call thread2

        wait

        writeln("Exiting main")
end
```

a. Hapus kode sebelumnya dengan cara: pada jendela CPU Simulator, klik REMOVE ALL PROGRAMS.
b. Compile kode diatas dan muatkan ke memori.
   Pada jendela compiler, carilah baris pada *program code (output*) yang menerjemahkan **synchronise,** catatlah kode asembler beserta keterangannya.
c. Pada jendela OS simulator buatlah proses baru, kemudian jalankan.
d. Tunggu hingga program selesai, sembari mengamati yang terjadi pada sub jendela waiting processes,
e. Catat dua nilai variabel g yang ditampilkan pada layar consol.

3. Kita modifikasi lagi kode sebelumnya. Bagian yang ditambahkan ditandai dengan tebal.

```
program CriticalRegion
        var g integer

        sub thread1 as thread
             writeln("In thread1")
             enter
             g = 0
             for n = 1 to 20
                  g = g + 1
             next
             writeln("thread1 g = ", g)
             leave
             writeln("Exiting thread1")
        end sub

        sub thread2 as thread
             writeln("In thread2")
             enter
             g = 0
             for n = 1 to 12
                  g = g + 1
             next
             writeln("thread2 g = ", g)
             leave
             writeln("Exiting thread2")
        end sub

        writeln("In main")

        call thread1
        call thread2

        wait
        writeln("Exiting main")
end
```

   a. Hapus program sebelumnya dari memori.
   b. Compile kode diatas, muatkan ke memori.
      Amati hasil kompilasi perintah enter dan leave pada jendela compiler output
   c. Beralihlah ke OS simulator, buat proses baru, kemudian jalankan.
   d. Catat 2 nilai variabel g.

4. Jadi apa kesimpulannya? Untuk memahami percobaan diatas, jawablah pertanyaan-pertanyaan berikut ini:
    a. Jelaskan tujuan utama praktimum ini menurut anda.
    b. Mengapa kita menggunakan variabel global (g) yang sama pada dua thread?
    c. Sudahkan kita menggunakan variabel lokal, dan apakah hasilnya akan berbeda? Lakukan eksperimen kecil dengan sedikit modifikasi pada kode yang ada dan jalankanlah program tersebut.
    d. Pada modifikasi yang pertama ditambahkan kata kunci **synchronise**. Jelaskan tujuan modifikasi ini. Beri contoh istilah untuk synchronise dalam bahasa pemrograman nyata.
    e. Pada modifikasi yang kedua digunakan kata kunci **enter** dan **leave.** Jelaskan fungsi modifikasi ini, dan apa bedanya dengan (d)?
    f. Critical regions seringkali diimplementasikan menggunakan **semaphore** dan **mutex.** Jelaskan pengertiannya dan apa perbedaannya.
    g. Berikan contoh nyata untuk suatu critical region (atau mutex region). Beri contoh nyata suatu mutex.
    h. Beberapa arsitektur komputer memiliki instruksi “test-and-set” untuk menerapkan critical region. Jelaskan bagaimana teknik ini bekerja dan mengapa penerapannya pada hardware.
    i. Jika hardware maupun OS tidak menyediakan bantuan, bagaimana anda dapat memproteksi critical region dalam kode anda? Berikan saran anda dan jelaskan dimana perbedaannya dengan metode-metode yang telah diuji coba kan diatas.

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM VII

Penjadwalan Proses 2 dan Manajemen Memori

A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Mendeskripsikan keadaan proses dan transisi akibat timeout.
2. Menjelaskan perbedaan antara penjadwalan pre-emptive dan non-pre-emptive.
3. Menelusuri nilai register PC dalam PCB suatu proses ketika berada dalam antrian Ready.
4. Menjelaskan pemanfaatan nilai Register PC pada penjadwalan Round Robin.
5. Menjelaskan page swapping ketika proses berpindah dari keadaan ready ke running

B. Dasar Teori

Pada materi kuliah dibahas mengenai berbagai jenis penjadwalan proses dan manajemen memori. Praktikum ini berdasarkan materi tersebut.

C. Langkah-langkah Praktikum

- **Penjadwalan**

1. Masukkan kode ini pada compiler :

```
program LoopForeverTest
     While true
          N = N + 1
     wend
end
```

a. compile kode diatas, kemudian muatkan ke memori.
b. Beralihlah ke jendela OS Simulator.
c. Pilih program LoopForeverTest, selanjutnya kita akan membuat 3 proses, namun sebelum tiap proses dibuat aturlah nilai LIFETIME dan pilih SECS untuk tiap proses: **10 seconds, 32 seconds, 6 seconds**.
d. Berikutnya memilih nilai timeslot melalui menu pull-down pada sub jendela SCHEDULER/POLICIES/RR Time Slice/Secs, pilih 4.
e. Tekan START dan tunggu hingga semua proses selesai.
f. Bukalah jendela OS ACTIVITY LOG dengan klik pada tombol VIEW LOG... Amati data log yang relevan kemudian perhatikan urutan pada proses yang berjalan (running processes) dan catat waktu yang dihabiskan oleh tiap proses selama dalam keadaan running.

2. Sekarang, beralihlah ke compiler dan masukkan kode berikut ini. Klik NEW untuk membuat tab editor baru.

```
program LoopForeverTest2
     while true
          n = n + 1
          i = i + n
          p = n + i + 5
     wend
end
```

   a. Compile kemudian muatkan ke memori kode diatas.
   b. Beralihlah ke OS Simulator.
   c. Pada program list akan tampak 2 program : LoopForeverTest, dan LoopForeverTest2.
   d. Klik **LoopForeverTest** dan buatlah satu proses.
   e. Klik **LoopForeverTest2** dan buat satu proses.
   f. Sekarang pada antrian ready seharusnya ada 2 proses.
   g. Pilih penjadwalan ROUND ROBIN, tipe prioritas NON-PREEMPTIVE dan RR Time Slice adalah **10 tick**.
   h. Atur pada setengah kecepatan simulasi.

Lakukan langkah-langkah percobaan berikut ini:

1. Sorot proses kemudian klik tombol PCB... amati nilai pada PC REGISTERS (pada PCB Info) pada masing-masing proses, dan catatlah.
2. Klik **START** dan siapkan mouse anda pada tombol SUSPEND (jangan klik).
3. Sewaktu running process kembali ke ready queue, segera klik tombol SUSPEND.
4. Sorot proses pada antrian ready, dan klik tombol PCB... pada antrian ready. Catatlah nilai PC REGISTER nya.
5. Sekarang, beri check pada SUSPEND ON RUN dalam tampilan RUNNING PROCESS. Kurangi kecepatan OS simulation. Klik tombol RESUME dan amati ketika proses yang antri kembali dalam keadaan running, yang menyebabkan simulasi **berhenti (suspend) secara otomatis**.
6. Perhatikan baik-baik pada kotak PC REGISTER dalam **jendela CPU Simulator**. Sekarang klik RESUME pada OS Simulator. Catatlah nilai pada PC REGISTER ketika nilainya berubah. Berikan pendapat anda terhadap nilai tersebut dan jelaskan apa yang terjadi.

- **Manajemen Memori**
  1. Masih menggunakan ROUND-ROBIN (RR) dan NON-PREEMPTIVE pada OS Simulator.
  2. Masukkan kode berikut ini pada compiler :

```
program ForeverLoop
     While true
          I = 1
     wend
end
```

     compile kode diatas, kemudian muatkan ke memori. Beralihlah ke jendela OS Simulator.

  3. Klik tombol VIEW MEMORY.... aktifkan STAY ON TOP. Matikan PAGING ENABLED (dengan menghilangkan tanda check). Pilih program ForeverLoop, buatlah proses dengan ketentuan berikut :

Proses	Page
P1	3
P2	5
P3	4
P4	2
P5	6

     Perhatikan 4 proses yang terdapat pada jendela READY PROCESS.
     Amati data pada kolom SWAP. Beberapa tertulis "Yes", jelaskan artinya.
     Bandingkanlah dengan tampilan *page* yang tampil pada jendela MAIN MEMORY.
  4. Klik VIEW UTILIZATION... informasi apa yang anda lihat? Klik STAY ON TOP. Sekarang, pada jendela OS Simulator, atur kecepatan pada kisaran 90, tekan START. Amati hal-hal berikut ini dan buatlah catatan:
     1. Jelaskan kejadian yang tampak pada jendela MAIN MEMORY (RAM).
     2. Apa yang tampak pada kolom SWAP di jendela RUNNING PROCESSES? Mengapa selalu sama? Apakah ada kalanya berbeda dengan yang ditampilkan pada kolom SWAP di jendela READY PROCESSES? Jelaskan.
     3. Amati bagaimana proses-proses yang ready berganti (swap) keadaan dari waktu ke waktu. Beri penjelasannya.
     4. Informasi apa yang anda peroleh dari jendela RESOURCE UTILISATION terkait dengan manajemen memori?

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM VIII

Deadlock

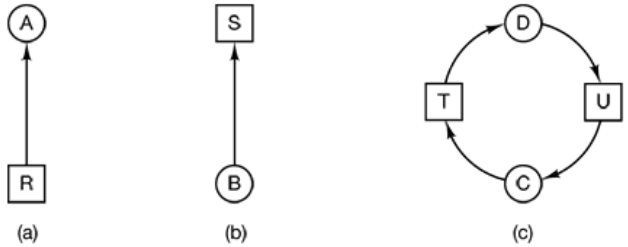
A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Menggunakan graf pengalokasi sumber daya (*resource allocation graph*) untuk mendeteksi deadlock.
2. Menggunakan simulator untuk membuat situasi deadlock.
3. Menggunakan dua metode untuk mengatasi deadlock

B. Dasar Teori

Deadlock yang terjadi ketika proses-proses membutuhkan akses ke lebih dari satu sumber daya (yang tidak dapat dipakai bersama) sering terjadi pada sistem operasi modern.



Keterangan:

a. Proses A sedang memakai (allocate) sumber daya R
b. Proses B meminta (request) sumber daya S
c. Proses C meminta sumber daya T yang sedang dipakai oleh D, namun D tidak (akan) melepas D karena sedang meminta sumber daya U yang dipakai oleh C.

OS dapat menerapkan berbagai teknik unuk menghindari, mencegah, atau mengatasi deadlock yang akan menyebabkan sistem berjalan se-efisien mungkin.

C. Langkah-langkah Praktikum

1. Pada kertas, buatlah suatu siklus *resource allocation graph* untuk 4 proses dengan 4 sumber daya (bisa sumber daya apa saja). Hanya empat sumber daya dengan tiap proses menggunakan satu sumber daya dan meminta sumber daya yang lainnya. Kondisi ini akan menyebabkan deadlock.
   a. Gambarlah graf untuk skenario ini dan jelaskan bagaimana deadlock dapat terjadi.
   b. Berapa proses yang mengalami deadlock untuk kasus ini?

Jalankan Simulator.

2. Tuliskan kode berikut ini pada compiler.

```
Program Deadlocks
     while true
          n = 1
     wend
end
```

   a. Kode diatas akan menjalankan loop selamanya.
   b. Compile program tersebut, muatkan ke memori.
   c. Beralihlah ke OS simulator.
   d. Sorot program Deadlocks pada PROGRAM LIST, klik tombol CREATE NEW PROCESS.
   e. Pastikan menggunakan ROUND ROBIN dengan 15 ticks.
   f. Maksimalkan kecepatan simulator. (**jangan klik START dulu**)
   g. Kemudian ikuti langkah-langkah berikutnya.

3. Simulator dapat mengalokasikan dan men-dealokasikan beberapa sumber daya secara imajiner untuk proses sehingga kita dapat mempelajari permasalahan terkait pengalokasian sumber daya dan deadlock.
   - Buatlah satu proses lagi sehingga akan ada dua proses pada antrian ready.
   - Biarkan berada pada antrian ready (**jangan klik START dulu**)
   - Klik tombol VIEW RESOURCES... pada jendela OS simulator, akan muncul jendela System Resources. Ada 4 sumber daya yang teridentifikasi yaitu R0, R1,R2, dan R3.
   - Gambar yang mewakili sumber daya berwarna hijau yang artinya sumber daya tersebut belum dialokasikan ke suatu proses.
   - Klik STAY ON TOP pada jendela resource.
   - Kemudian pilih proses pada antrian ready dan klik tombol *allocate*.
   - Akan tampak nomor ID proses pada kotak USED BY dan warna sumber daya akan berubah menjadi kuning.
   - Kemudian klik (ID) proses yang lain dan lakukan hal yang sama, maka nomor ID akan tampak dalam kotak daftar *requested by,* dan warna menjadi merah. Jelaskan yang terjadi.
   - Berlatihlah membuat kondisi deadlock untuk 2 sumber daya.
   - Sekarang, klik tombol RELEASE ALL untuk men-dealokasikan sumber daya. Sumber daya seharusnya menjadi hijau kembali dan nomor ID proses hilang.
   - Hapus proses pada antrian ready. Untuk melakukannya, klik tombol CLEAR pada jendela READY PROCESSES.

4. Setelah memahami mekanisme pengalokasian sumber daya menggunakan simulator, berikutnya buatlah 4 proses yang berkaitan dengan graf alokasi sumber daya yang tadi anda buat. Kemudian alokasikan sumber daya tersebut untuk menciptakan kondisi deadlock. Bagaimana hasilnya? Jika berhasil, apa yang anda lihat?

5. Sekali deadlock terbentuk, cobalah menjalankan proses dengan klik pada tombol START. Jelaskan yang terjadi. Proses pada antrian ready belum mengalami deadlock, kejadian tersebut terjadi ketika proses telah berjalan.

6. Dalam kondisi proses yang deadlock, apa yang dapat anda lakukan untuk mengatasinya? Berikan dua cara untuk mengatasinya, kemudian praktikkan menggunakan simulator. Beri penjelasan terhadap hasil simulasi.

MODUL PRAKTIKUM SISTEM OPERASI

PRAKTIKUM IX

I/O INTERRUPT

A. Tujuan

Pada akhir praktikum ini, peserta dapat:

1. Menjelaskan pengertian interrupt handler.
2. Menjelaskan mekanisme I/O interrupt handler.
3. Menggunakan simulator untuk menghasilkan console input interrupt.

B. Dasar Teori

Terdapat 3 teknik yang dapat digunakan dalam operasi I/O:

1. I/O terprogram

   Pada I/O terprogram data saling dipertukarkan antara CPU dengan modul I/O. CPU mengeksekusi program yang memberikan operasi I/O kepada CPU secara langsung. Ketika CPU mengeluarkan perintah ke modul I/O, maka CPU harus menunggu sampai operasi I/O selesai. Apabila CPU lebih cepat dibandingkan modul I/O maka hal ini akan membuang-buang waktu CPU.

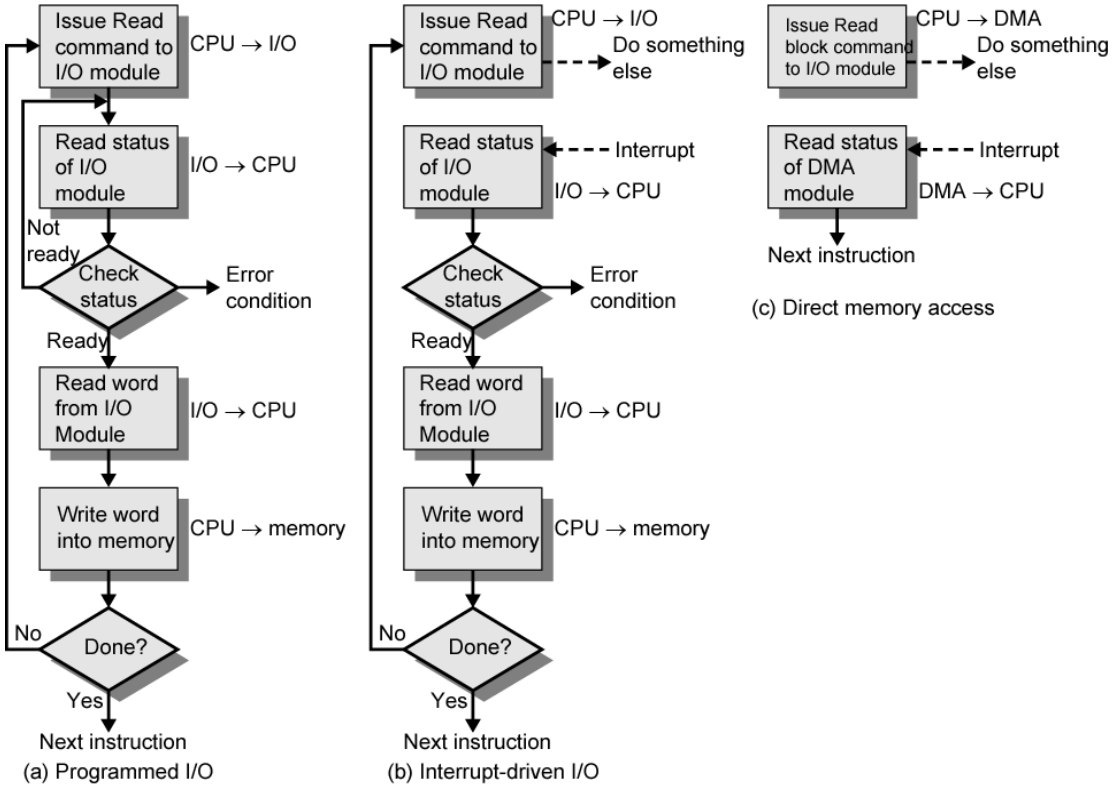
   Permasalahan (busy waiting) ini dapat diterima untuk beberapa kasus, misalnya pada embedded system karena tidak ada tugas lain bagi prosesor untuk dikerjakan.
2. I/O Interrupt

   Dengan *interrupt driven I/O*, CPU mengeluarkan perintah I/O, kemudian dapat meninggalkannya untuk mengerjakan pekerjaan yang lain. Apabila perintah I/O tersebut telah selesai dilaksanakan oleh perangkat yang bersangkutan maka modul I/O dapat menginterupsi CPU.
3. Direct memory access (DMA)

   Baik menggunakan I/O terprogram maupun I/O interrupt, CPU bertanggung jawab atas pengeluaran data dari memori utama untuk keperluan output, maupun penyimpanan data di dalam memori utama untuk keperluan input. Sebagai alternatif, digunakan *direct memory access* yang memungkinkan modul I/O dan memori utama dapat saling bertukar data secara langsung, tanpa melibatkan CPU.

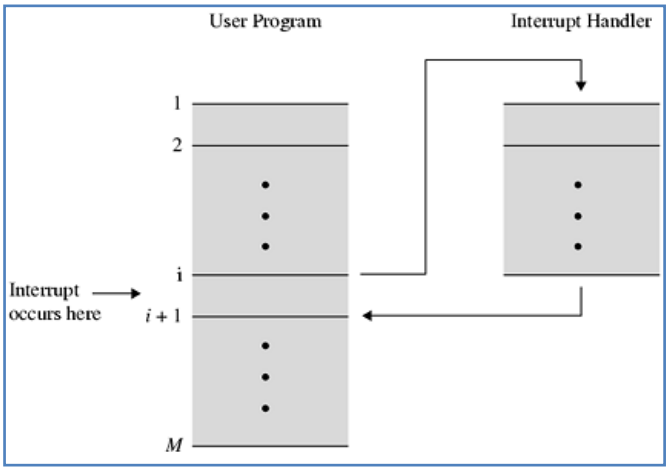
	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Gambar 1. Rangkuman tiga teknik operasi I/O



Gambar 2. Diagram alir tiga teknik operasi I/O

Pada praktikum ini kita menitikberatkan perhatian pada I/O interrupt dan mengkaji mekanisme I/O interrupt handler. Interupsi digunakan untuk beralih dari suatu proses yang sedang berjalan ke **alamat awal kode *sub-routine*** yang berada pada lokasi khusus yang telah diketahui oleh CPU. Jika nilai alamat tersedia ketika terjadi interupsi tertentu, maka CPU mulai mengeksekusi kode sub-routine (*interrupt handler*) mulai dari alamat awal tersebut.



Gambar 3. Pengalihan kontrol ketika terjadi interrupt

C. Langkah-langkah Praktikum

Jalankan Simulator.

1. Tuliskan kode berikut ini pada compiler.

```
program IONoInterrupt
      n = 0
      i = 0
      while i <> 13
            read(nowait, n)
            if n > 0 then
                  i = n
                  writeln("You entered key code : ", n)
                  n = 0
            end if
      wend
end
```

Program diatas membuat program bernama IONoInterrupt. Program ini menerima **kode-kode** yang dikirim dengan menekan keyboard kemudian menampilkannya pada konsol. Program tersebut akan terus mengulang proses hingga menerima kode 13 (desimal). Ketika menerima 13 maka program berhenti.

- Disebut apakah kode-kode tersebut? Bersesuaian dengan tombol keyboard apakah kode 13?
- Apa tujuan penggunaan kata kunci "nowait" (pada statement read)?

Compile, muatkan ke memori, beralihlah ke **OS Simulator**.
Buatlah proses dari program **IONoInterrupt**. Gunakan penjadwalan **FCFS**, posisikan kecepatan simulasi pada **maksimum**. Ikuti langkah-langkah berikut :

a. Pada jendela **CPU Simulator**, klik tombol **INPUT/ OUTPUT**, aktifkan Stay on top, serta klik tombol **SHOW KBD**... untuk menampilkan keyboard.
b. Kembali ke jendela **OS Simulator**, tekan START.
c. Amati bahwa pada **CPU Simulator** akan tampak cpu terus mengeksekusi program (loop). Klik salah satu tombol **keyboard** dan amati **konsol**. Apa yang ditampilkan?
d. Ulang beberapa kali memberi input dari keyboard tersebut.
e. Klik tombol "Enter". Apa yang terjadi?
f. Pada jendela CPU Simulator, klik tombol **INTERRUPTS**... kemudian catat jika ada nilai pada kotak teks.

2. Sekarang modifikasi kode sumber sebagai berikut (klik *new* pada editor compiler).

```
program IOInterrupt1
      sub inth intr 1
            write("You entered key code : ")
      end sub

      n = 0
      i = 0
      while i <> 13
            read(nowait, n)
            if n > 0 then
                  i = n
                  writeln(n)
                  n = 0
            end if
      wend
end
```

   a. Compile, muatkan ke memori, jalankan pada OS Simulator.
   b. Apakah ada perbedaan perilaku dari versi (1) sebelumnya? Jelaskan.
   c. Pada jendela CPU Simulator, klik tombol INTERRUPTS... dan buat catatan nilai pada kotak teks. Apa perbedaan dengan versi (1)? Jelaskan arti penting perbedaan tersebut dan bagaimana penggunaannya.
   d. Beralihlah ke Jendela compiler. Amati pada alamat mana **sub inth** dimulai? Apakah nilainya sama dengan **INT1** (pada jendela **Interrupt** CPU Simulator**)**?

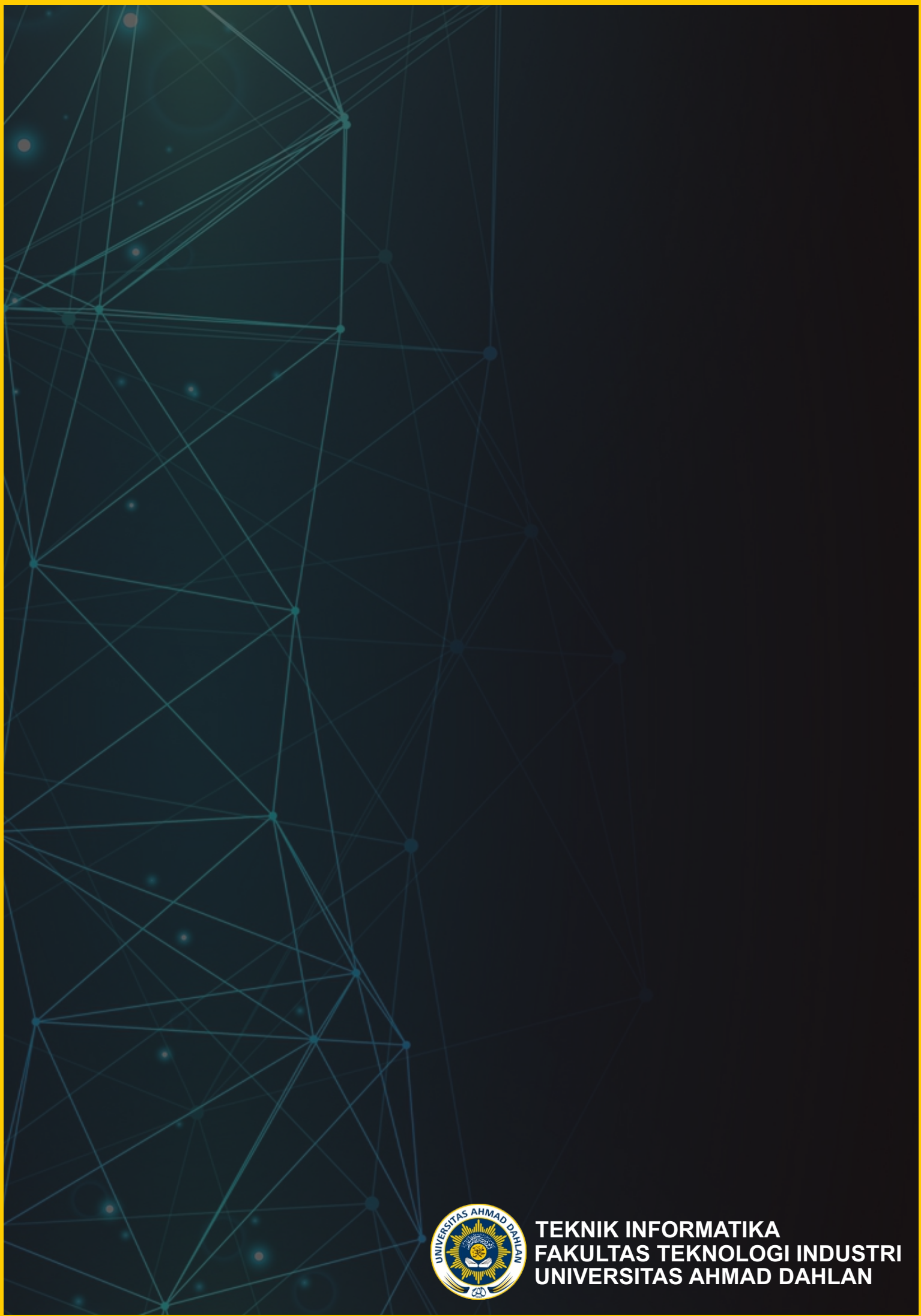
3. Sekali lagi modifikasi kode sumber sebagai berikut (klik *new* pada editor compiler).

```
program IOInterrupt2
      sub inth intr 1
            write(", You entered key code : ")
      end sub

      n = 0
      while n <> 13
            write("Press a key : ")
            read(n)
            writeln(n)
      wend

      writeln("End of program")
end
```

   a. Compile, muatkan dalam memori, dan jalankan pada OS Simulator.
   b. Apakah ada perbedaan perilaku dari versi (1) sebelumnya? Jelaskan (**Hint**: untuk mengetahui pengaruh kode `no wait`, anda perlu mengamati jendela INSTRUCTION MEMORY pada CPU Simulator, dan jendela WAITING PROCESSES pada OS Simulator).

4. Sekarang modifikasi kode diatas dengan mengganti nilai kata kunci **intr** pada **sub inth** menjadi **2**.
    a. Compile, dan jalankan program yang telah dimodifikasi pada OS Simulator.
    b. Mengapa hasilnya berbeda dengan versi sebelumnya? Jelaskan.

Jendela Interrupt menampilkan nilai **kode alamat** bagi berbagai tipe penyebab interrupsi (*console input, timer, exception, software, user1,* dan *user2*). Jika nilai alamat tersedia, maka kode-kode pada alamat tersebut akan dieksekusi saat terdeteksi interrupt yang bersesuaian (misalnya: *console input*); jika tidak ada nilainya maka interrupt diabaikan.



TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS AHMAD DAHLAN