Subject:

Year. Month. Date. ( )

26

چندین روند که در اجرا داشته باشیم ← multi Tasking

چه مکانیزمی برای هماهنگی بین Task ها استفاده می کنند ← 1. وقفه مثال تولیدکننده مصرف کننده

2. سمافور

3. Guarded statement

1. وقفه:

مشکلات استفاده از وقفه برای همزمانی:

1. برنامه منتظر دریافت وقفه باید طوری باشد که بتواند در هر لحظه وقفه را بپذیرد یا باید در هر لحظه آمادگی پذیرش و پردازش صحیح آن را داشته باشد

2. برنامه منتظر دریافت وقفه باید یک busy loop را اجرا کند

3. کد int handler به قسمتهای از برنامه امکان اجرا دارد ← که بستگی به زمان اجرای برنامه

4. " " جزئی از Task به حساب نمی آید.

سمافور:

semaphore یک data object است شامل یک صف و یک شماره و دو operation (wait و signal)

← اگر wait روی semaphore p در برنامه A اجرا شود به شمارنده‌ی P رجوع می شود اگر شمارنده غیر صفر بود یک واحد کم می شود سپس A به کارش ادامه می دهد و اگر شمارنده صفر ← A در صف منتظران

← اگر signal روی semaphore P در برنامه‌ای اجرا شود به صف منتظران P رجوع می شود در صورتیکه Task های در این صف باشند یکی از آنها خارج می شود یک واحد به شمارنده اضافه می شود و اجرای A ادامه می یابد اگر صف خالی باشد

برای ناحیه بحرانی

```
تولیدکننده                         مصرف کننده
task A                             task B
  :                                  :
begin   → تولید اطلاعات              begin
  loop                                 loop
   signal (p)                           wait(p)
                                        مصرف اطلاعات
   wait (Q)                             signal(Q)
  endloop                              endloop
end                                  end
```

Subject:
Year. Month. Date. ( )

~~task A~~ بازگشت

تعداد اولیه صف Q صفر است

A فراهم شده اطلاعات و B مصرف کننده آن. همزمان B اطلاعات تولید شده را مصرف کرد A می تواند مجدداً تولید نماید

۱. استفاده از چنین سمافور منطق برنامه را پیچیده می کند

2. اگر task به دلیل operation نتواند به درستی عمل کند Deadlock پیش بیاید

3. برنامه به طور کلی منتظر یک semaphore می ماند حال اگر شاید لازم باشد منتظر سیگنال دیگری باشد

Guarded statement در زبان Ada

دستور select در زبان ada

```
when (condition1) => statement 1
or then ( cond 2) => statement2
        :
or when (cond n) => statement n
else statement n+1
end
```

دستور شبیه else if است با این تفاوت که اگر چندین شرط همزمان برقرار بود به صورت تصادفی یکی را انتخاب می کند و عبارت معادل را اجرا می دهد

در زبان Ada برای همزمانی بین Task ها از تکنیک قرار ملاقات استفاده می شود به این ترتیب که اگر دو task از نوع A و B بخواهند همگام شوند کد آن

```
task A              taskB
Ready 1;            accept ready do
                        عبارت
                    end;
```

PAPCO

Subject:

Year. Month. Date. ( )

مثال برای Scope

```
procedure main ( );
Begin main
    procedure P(x)
    Begin P
        procedure Q(a,b)
        begin Q
            R(x,y);
        end Q;
    end P;
    Procedure R(c,d)
        => x مرجع به
    end R
    P(w);
    R(u,v)
End main;
main( );
```

87 ,8 ,28

Data Control

چگونه بین زیربرنامه‌ها، برنامه‌های مختلف چگونه به data های مختلف ارجاع دهیم.

Formal par ← وقتی تابع تعریف می‌کنیم ورودی‌هایی با اسم‌هایی که به کار می‌بریم در اصل اسم متغیر نبوده

association ← هایی اجرا شده در حین قطع می‌شود.

1. در شروع اجرا ← association های بین اسامی و data object ها و اسم زیربرنامه‌ها و تعریف آن‌ها ایجاد می‌شوند

2. در حین اجرا ← reference Operation ها برای بازیابی Data Object یا زیربرنامه متناظر

3. فراخوانی یک زیربرنامه‌ای دیگر توسط برنامه ← در شروع اجرای زیربرنامه association های جدید بین اسامی data object ها/ اسم پارامترها و Data Object متناظر آن‌ها و اسم زیربرنامه و تعریف آن زیربرنامه ایجاد می‌شود

4. Reference Operation ← برای بازیابی data object صحیح و زیربرنامه صحیح متناظر می‌شوند

5. پایان اجرای زیربرنامه ← association های مربوط آزاد می‌گردد و به محیط برنامه اصلی برمی‌گردیم

محیط‌های ارجاع ← اگر متغیر local باشد در محیط آن تابع association باشد

1. محلی association هایی که در زیربرنامه ساخته می‌شود
2. association هایی که در سایر زیربرنامه‌ها ایجاد شده‌اند دیگر برنامه می‌تواند از آن‌ها استفاده کند
3. association هایی که در برنامه اصلی تعریف شده‌اند و این برنامه و سایر برنامه‌ها می‌توانند از آن استفاده کنند
4. association از پیش تعیین شده

Subject:

Year. Month. Date. ( )

مثال: انواع محیط‌های ارجاع برای Sub2

محیط ارجاع محلی: D, C

" " غیرمحلی: B, Sub2, A, Sub1,

محیط ارجاع خارجی: B, Sub1,

انواع محیط‌های ارجاع برای Sub1

محلی: A, B, sub2

غیرمحلی: Sub1, B, C

خارجی: Sub1, B, C

انواع محیط‌های ارجاع برای main

محلی: Sub1, A, B, C

غیرمحلی: " " " "

خارجی: " " " "

← Visibily

← slide 10

I is visible, I is not	Sub1
I visible, J is not	Sub2
Neither is visible	main

← slide 11

کلید محیط اجرا: کلیدی که چند اسم را دنبال می‌شود

PAPCO

Subject:
Year. Month. Date. ( )

```
A1                      A2                      A3
:                       :                       :
Ready1;                 Ready2;                 Ready3;

task B
:
select
   accept Ready1 do ... end
   or accept Ready2 do ... end
   or accept Ready3 do ... end
   ~~or accept~~
end
```

در این حالت هر دو task که به نقطه قرار ملاقات برسند، B انتخاب می‌کند به صورت همزمانی اجرا می‌شوند

Real Time Application وابسته به زمان بوده و بعضی از زمان‌ها اگر جواب یک الگوریتم دیگر جوابش به درد نمی‌خورد

```
task A                                   task B
  Select                                 accept ready1 do
     Ready1;          →  0.5 ثانیه وقت برای اجرا دارد          :
     or delay 0.5;    →  0.5 ثانیه طول بکشد رها شده است       end
  end;
```

اگر بعد از 0.5 ثانیه این کار انجام نشود سیستم کار دیگری انجام می‌دهد

در زبان Ada برای کنترل Task هایی که کارشان از اهمیت ویژه‌ای برخوردار است از Guarded statement ها استفاده می‌شود

Subject:
Year. Month. Date. ( )

کنترل ترتیب اجرا: ← expression
statement
subroutine

محاسبه‌ی عبارت:

Lazy Evalution:

Eger Evalution → $(A+B) * (A+D)$

محاسبه را به عملگر بگذار یعنی عملگرها fetch ... $Z + (y = 0 ? x : x/y)$

می‌شوند ولی محاسبه خود عملگر است. یعنی اگر انجام شد دعلی است error بدهد.

بعضی از عملگرها محاسبه‌ی عبارت را اجرا نمی‌شود ـــ یک برنامه‌ای در پشت صحنه باشد که در حین اجرا اجرا خواهی شود.

محاسبات Lazy بیشتر در زبان های functional مطرح می‌شوند

در زبان های object-oriented از تکنیک Eager استفاده می‌کنند به خاطر عملکرد بیشتر است

جلسه‌ی دهم:

ترتیب اجرای statement ها:

ـ به صورت ترتیبی پشت سر هم (composition).

ـ اجرای برنامه های کاربردی، مشتمل بر اجرای زیر برنامه هست.

همیشه بهتر نیست که همه‌ی دستورات اجرا است.

Goto را استفاده نکنید چون: خوانا نیست (در برنامه خوانایی وجود ندارد)

سرعت اجرای برنامه را پایین می‌آورد.

درحقیقت abstraction را از بین می‌برد.

(AMD برای کسب برنامه‌نویسی قدیمی معتبر است، و میزان cache در CPU ها می‌تواند اجرای برنامه را تضمین کند)

مدیریت حافظه ـــ صفحه‌بندی

PAPCO

Subject:
Year. Month. Date. ( )

کل برنامه را به RAM نمی‌آوریم و آن جایی را می‌آوریم که می‌خواهیم اجرا کنیم و بقیه در قسمتی از hard که حافظه مجازی می‌گوییم و مشخص است کجاست ذخیره می‌کنیم

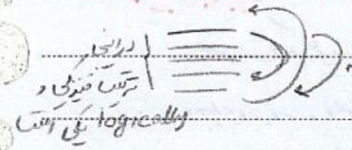
goto به این علت برنامه را کند می‌کند که الان قسمتی از برنامه که goto داخل آن است داخل RAM است ولی ممکن است آدرس جایی goto در جایی باشد که در داخل RAM نیست باید آن قسمت را به RAM منتقل کنیم

محاسن goto

- با کمک آن ادبیات زبان ساده می‌شود (پرکنش بیشتر است)
- بسیار توانا است (دستورات کنترلی به صورت goto ترجمه می‌شوند) ← دستورات انتقال کنترل بیان پیاده می‌شوند
- کاربرد آن آسان است
- برای برنامه‌نویسان assembly تداعی‌کننده انتقال کنترل است
  ↓
  while / if / for

معایب goto

← به برنامه flat structure می‌دهد ← یکجا را نمی‌بینیم و برنامه اصلاً قابل فهم نیست ← اگر قرار باشد برنامه را ببینیم در کجای اجرای برنامه ممکن است function باشد ...
← دستورات براساس ترتیب اجرا می‌شوند spaggeti Logic
← مجموعه statement های بی‌چنین حدث را دنبال می‌کنند
روتین های چند منظوره ← برنامه از حالت پیوستگی پشت سرهم برای اجرای دستورات خارج می‌شود



براساس ترتیب فیزیکی پشت سرهم اجرا می‌شوند

1.point دستورات بعد از goto شرطی در optimizer احتمالاً حذف می‌شود
وظیفه‌اش این است که دستوراتی را که اجرا نمی‌شوند را حذف کند

- ممکن است goto عملش سنگین باشد و از 1000 خط رد بپرد چون هیچ قاعده و اصولی ندارد.

Subject:
Year. Month. Date. ( )

برنامه‌های ساخت یافته

- طراحی برنامه به صورت سلسله مراتبی ← abstraction

عبارات شرطی:

1) if condition then statement

2) " " " state 1 else state 2

3) " " 1 " "
else if " 2 " state2
else if " 3 "
:
else statement n

پیاده‌سازی دستور case

Case Tag
0 : begin st0 end
1 : " st1 "
2 : " st2 "
3 : " st3 "
otherwise begin st4 end

صورتی که در حالت پیاده‌سازی می‌شود استفاده از Jumptable است

fetch tag مقدار	$L_0 \rightarrow S_0$
$L \rightarrow$ L+t پرش به	پرش به L5
L_0 پرش به	$L_1 \rightarrow S_1$
L_1 پرش به	پرش به L5
L_2 " "	$L_2 \rightarrow S_2$
L_3 " "	پرش به L5
L_4 " "	$L_3 \rightarrow S_3$
	پرش به L5
	$L_4 \rightarrow S_4$
	$L_5 \rightarrow$

PAPCO

Subject:
Year.   Month.   Date.   ( )

حلقه‌ها:

اغلب زبان ها کامپایلر یکبار تعداد تکرار را ارزیابی کند و دیگر کاری به تغییرش ندارد.

در صورت غلط بودن متغیر کنترلی:

می‌توانیم error بدهیم یا اجرا نکنیم یا یکبار اجرا کنیم

حلقه‌های بی‌پایان:

```
loop
    Exit when شرط
end
```

از کل برنامه خارج می‌شود.

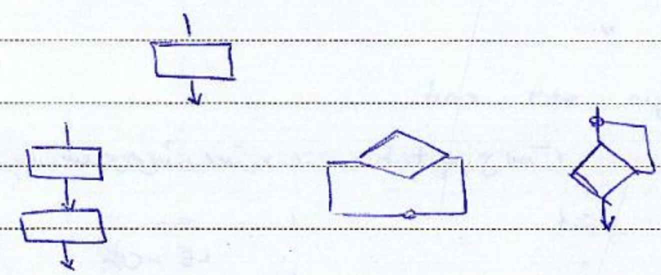
می‌توانیم هر برنامه‌ای طوری بنویسیم که از goto استفاده نشود

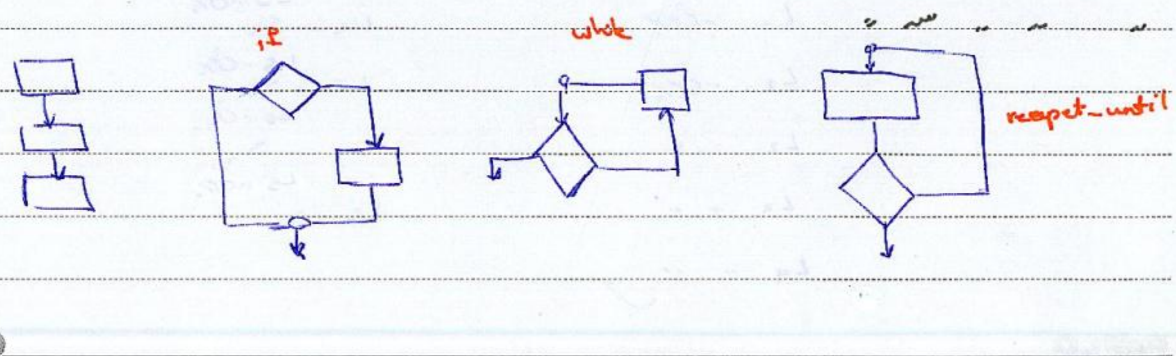
که برنامه‌نویسی سخت و حتی گاهی نیاز به goto پیدا می‌شود بهتر است استفاده از آن کم شود.

Proper programmer ← یک flowchart که یک پیکان ورودی و یک پیکان خروجی دارد.

به هر جا بتوانیم ببریم   که هر مسیری که می‌آید

به آن Prime یک node





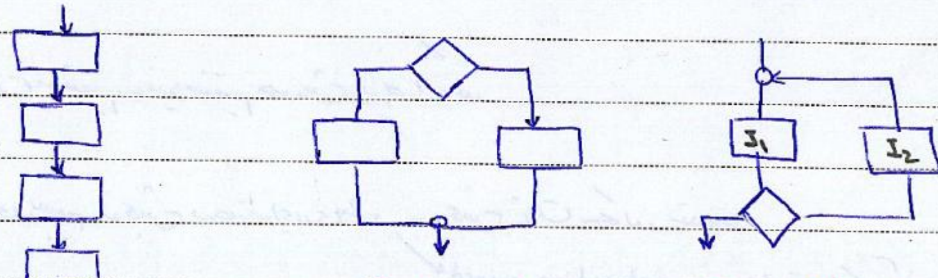
PAPCO

Subject:

Year. Month. Date. ( )

مواردی که کارهایی انجام نمی‌دهند به آن‌ها تابع همانی می‌گویند

برنامه‌های Prime با چهار node :



if then else   Do...while...Do

```
for i=1 to k do
      if vec[i]=0 then goto α
end for
```

87/8/14

در برنامه‌نویسی ساخت‌یافته سعی می‌کنیم از goto کمتر استفاده کنیم.

```
loop                ≡        loop for ever  معادل با
read (x)
if (x=0) then goto α
process (x)
end
```

(... exception

که شرایطی در برنامه پیش می‌آید که مجبور می‌شویم یک روتین را اجرا کنیم

در exception که خطایی پیش می‌آید سیستم یک goto دارد که اجتناب‌ناپذیر است

که گریز از آن بدون استفاده از goto خیلی کم است

PAPCO

Subject:

Year. Month. Date. ( )

کنترل اجرایی زیربرنامه‌ها:

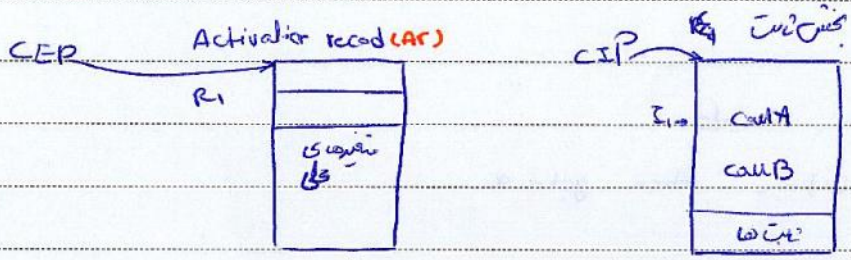
معمول‌ترین نوع ← call & Return

پیش‌فرض‌ها:

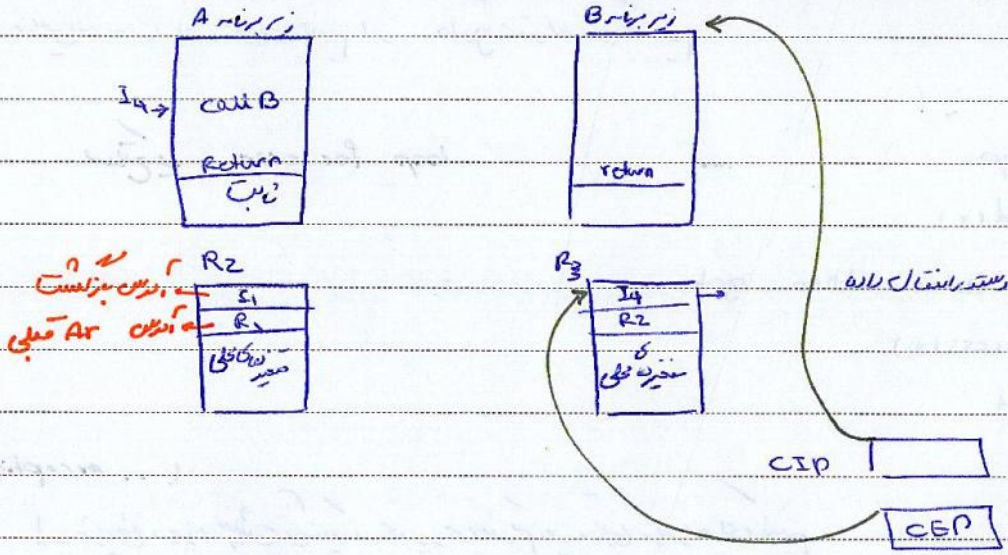
1. چیست مستقیم و یا غیرمستقیم خودش را فراخوانی کند

وقتی یک برنامه فراخوانی می‌شود ← بخش ثابت کد ساخته می‌شود

پارامترهای که بهش ارسال می‌شود ← activation record



در AR جاهای خالی برای ذخیره CIP، CEP وجود دارد



در Recursive باشد در DS، Activation record ذخیره می‌شود. پیاده‌سازی این روش‌ها با کامپایلر توسط Stack هم باشد.

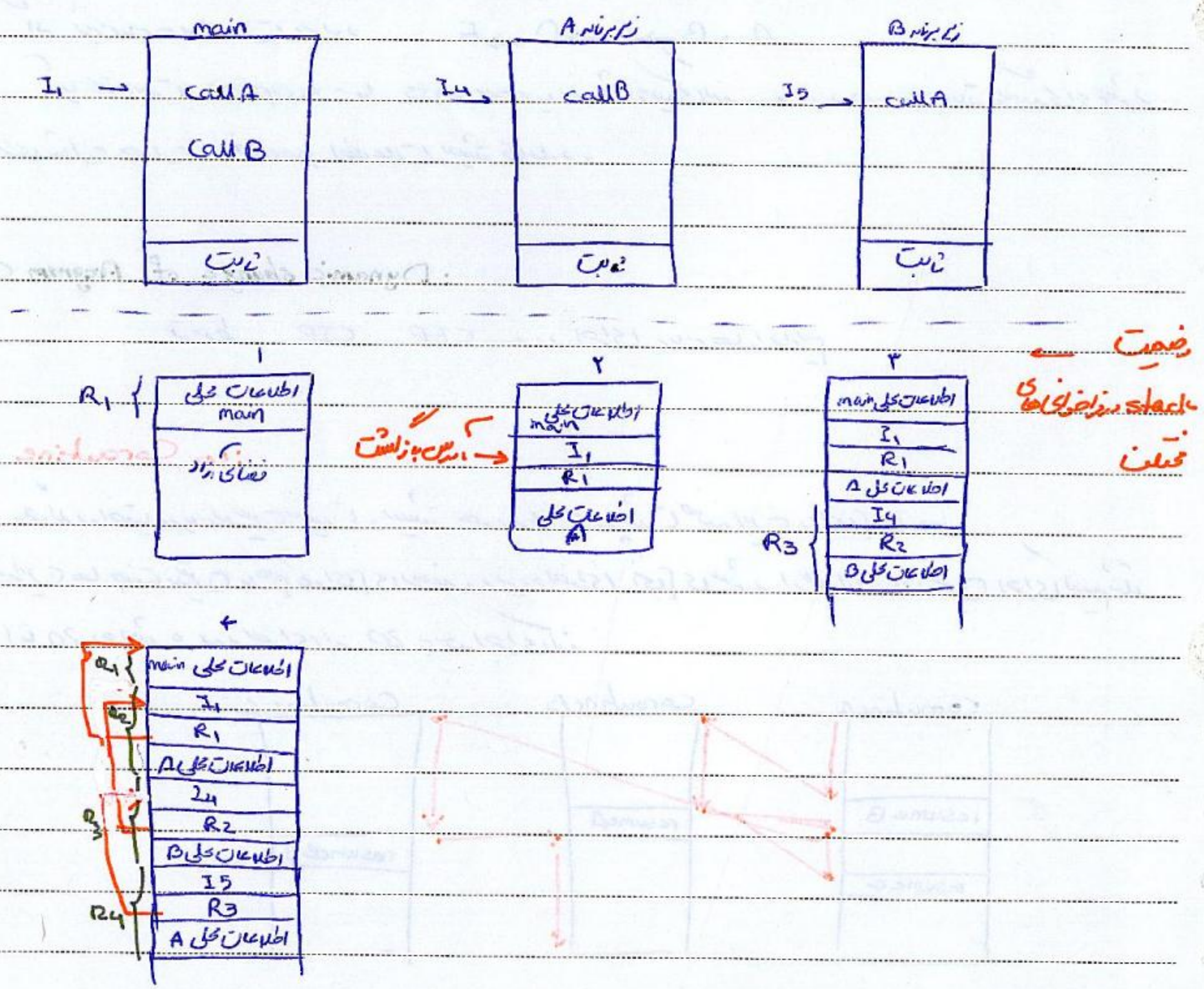
Subject:
Year. Month. Date. ( )

اول مشکل باید جای R_1 را پیدا کنیم و از این طریق بازگشتی را داشته باشیم

به ازای هر اجرای بازگشتی متغیر محلی باید بدهیم و پیدایش کنیم ← اگر ثابت و AR را هم می‌چسباندیم می‌دانیم که پشت سر هم قرار دارد.

activation record می‌تواند متغیر باشد ← بستگی به پیاده‌سازی دارد

در چسباندن به هم جدول هر کدام یکی گرفته می‌شود و ممکن است هدر رود



exception به وسیله ← تحت ابزار (رضایی مثل خطای عادی)

OS (برنامه را از کلیم خود برابر می‌گذارد یا error را خودش برطرف می‌کند

compiler

← به برنامه‌نویس اجازه می‌دهد exception handler بنویسید

جدا از پیگیری به exception برنامه‌های دیگر رسیدگی کند. آن‌ها اتفاق افتاده

اگر زیربرنامه‌ها به صورت زیر بود A → B → C → D → E

آیا بهتر است exception را pass بدهد یا خودش رسیدگی کند ← در درجه‌های بالاتر رسیدگی می‌شود

برای مثال چون parent اطلاعات بیشتری دارد.

Dynamic change of Program Calls:

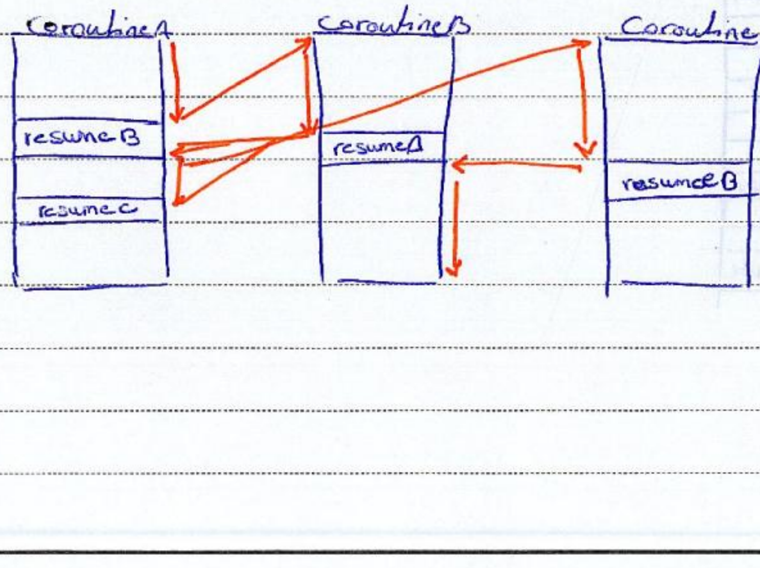
توسط CEP , CIP ← رد اجرای برنامه‌ها را داریم

Coroutine ها:

معمولاً در اختیار برنامه‌نویس نیست و بیشتر خود زبان در پشت صحنه این را انجام می‌دهد.

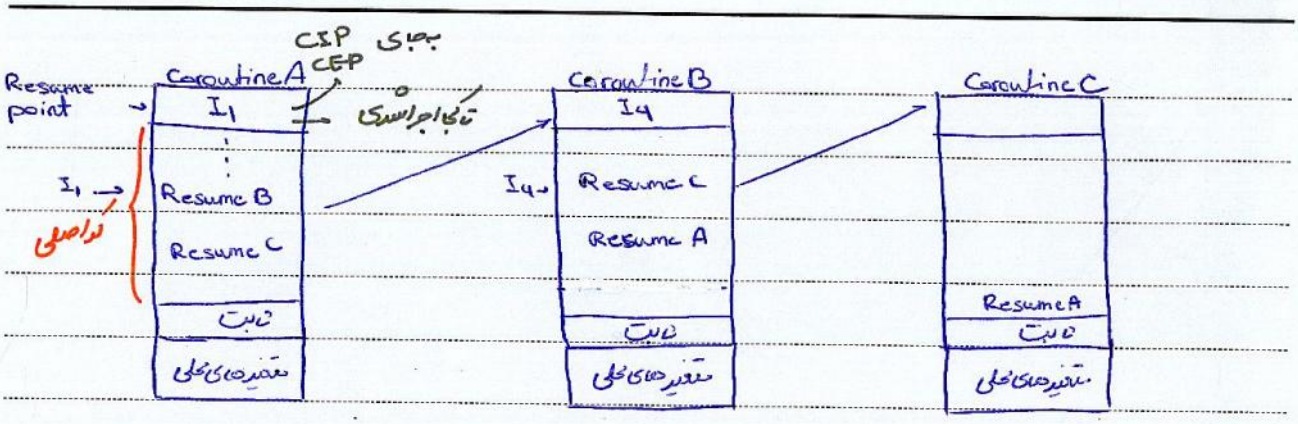
در این‌ها چند تا روتین با هم کار انجام می‌دهند و وقتی در اجرای آن‌ها می‌شود از همان ادامه‌ی روتین اجرا کنیم مثلاً

اگر C2 اجرا شده بعد از اجرای از C2 به بعد اجرا می‌کند



Subject:

Year. Month. Date. ( )



Resume point کجا است؟ activation record است.

Subject:

Year. Month. Date. ( )

جلسه ۱۹ ، ۸ ، ۸۷

کنترل ترتیب اجرا

✓ در عبارات

✓ در جملات

✓ در زیربرنامه ← call & Return

Recursive

Exception

Coroutine

Schedulers

Tasks

تفاوت scheduler: زیربرنامه معمولی هر وقت فراخوانی شد اجرا می‌شود ولی در scheduler این طوری نیست. مثل اینکه یک سری زیربرنامه داریم که باید پشت سرهم اجرا شوند.

✓ اجرا قبل یا بعد از سایر زیربرنامه‌ها — call A {after / before} B

✓ اجرا بر شرط عبارات منطقی — call A when x = 0

✓ اجرا بر مبنای مقیاس زمانی — call A at time = B

✓ اجرا بر اساس اولویت — call A with periority = 7;

سرویس برنامه‌ای که در background همواره در state اجرا است که interface ندارد و بخشی از windows است.

Subject:
Year. Month. Date. ( )

Task ها:

اگر یک برنامه در سیستم یک روند اجرای مشخص داشته باشیم — Sequential

Prog 1 Prog 2

با هم شروع به کار می‌کنند مثل کاری که OS انجام می‌دهد.
در نتیجه این زیربرنامه‌ها از طریق ترتیب اجرا تفاوت دارند

مفهوم جدیدش این است. Task ≡ Process

✓✓ اصطلاحات)) :

فرق multi tasking و multithreading :

multithread — یک برنامه است که خودمان نوشتیم می‌گوییم چند تا روند اجرای متمایز دارد که می‌توانند با هم اجرا شوند. اصطلاحاً threadهای جدا هستند و اگر اجرا نشوند کل فرآیند را از کار نمی‌اندازند. هماهنگی مربوط به فرآیند است در اختیار برنامه‌نویس است و خود برنامه‌نویس باید مراقب باشد

multi Tasking — فرآیندها هماهنگی برای هم‌روندی داشته باشند و خود سیستم این مسئله را کنترل می‌کند.
که چند تا process داریم که مدیریت این‌ها بر سطح بالاتر آن‌هاست
که چند تا thread داخل processها می‌شوند و مدیریتش با process است.

Distruted system ≠ parallel System

DS — مسئله را می‌شکنیم که چند تا کامپیوتر در جاهای مختلف انجام می‌دهند و نتایج را با هم یکی می‌کنند. نتیجه اصلی را به ما می‌دهند. از زبان‌های مشترک کامپیوترها در این‌ها برای محاسبات بهره می‌برند. آخر کامپیوتر مرکزی نتیجه‌ها را یکجا جمع می‌کند.

PS — مثل serverها، cpuها یک جا متمرکز است

حالت اجرای تک پردازنده‌ای: که چندین پردازشگر شبیه‌سازی می‌شود.

Subject:
Year. Month. Date. ( )

زبان‌های برنامه‌سازی parallel باید چه پیش‌نیازهایی داشته باشند

اصول زبان‌های برنامه‌سازی موازی:

mutable: متغیرهای معمولی (حالت عادی declare شدن و استفاده از آن‌ها مثل قبل است)

Definitional: (متغیرهای تعریفی) که فقط یکبار مقدار می‌گیرند (مقداردهی می‌شوند)
دسترسی همه Taskها به صورت صحیح و مقداری که وجود دارد مقدار ناخواسته‌ای نداشته باشد) به آن‌ها
صورت بپذیرد.

اضافه بر اجرای ترتیبی، اجرای parallel هم می‌خواهیم
که دستوری برای نشان دادن یک Task موازی باشد.
دستور parallel ← مانند fork در زبان C

1.3 الگوی Transformational: ورودی می‌گیریم و عملیات محدود تعریف روی آن انجام می‌شود و خروجی‌های مناسب
را به ما می‌دهند مثل ضرب ماتریس‌ها

2.3 الگوی Reactive: واکنش به رویداد خارجی مثل realtime system
realtime System سیستمی که در مدت زمان معینی پاسخ می‌دهد اگر نه system ، fail می‌شود
Transacting Processing : مثل سیستم رزرو بلیط قطارهای مسافری
Operating System.
که مشخص نیست در این‌ها رویداد چه زمانی اتفاق می‌افتد

IPC ← Inter Process Communication

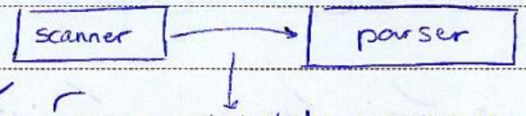
ارتباطات موازی ← حافظه مشترک
ارتباطات موازی ← صدور پیام (message passing)

Subject:
Year. Month. Date. ( )

Synchronization ← دو تا فرآیند می‌خواهند با هم کار کنند

تولیدکننده ← مصرف‌کننده

در پیاده‌سازی کامپایلر ← scanner و parser



scanner، data را برای parser آماده می‌کند

اجرای همزمان: دستوری به اسم and ← اجازه می‌دهد statementهای مختلف را به صورت همزمان (پیش‌بینی شده) اجرا کند

statement 1 and statement 2 and ... and statement n;

```
call ReadFromkeyboard and
 "   write Process     "
 "   Executeprocess    ;
```

} تا این سه برنامه اجرا نشوند به سراغ دستور بعدی نمی‌رود در حین CPU بین آن‌ها پاس می‌شود

این گزینه‌ها در زبان‌های سطح بالا درست در زبان‌های Ada، PL/I، C ← folk وجود دارد.

نحوه اعلان Task در Ada:

```
Task fi is
Task اعلان‌های خاص        } body of header
     برای ایجاد همزمانی   }
  end;
Task body fi is
Task اعلان‌های عادی برنامه برای اجرای
    begin
       دستورات

    end;
```