ساختار کنترل ترتیب اجرای دستورات به سه دسته تقسیم می شود:

1. **داخل عبارات (داخل دستوری)**: ساختار استفاده شده در محاسبات عبارات ریاضی
2. **بین دستورات:** ساختارهای استفاده شده در بین دستورات یا گروهی از دستورات مشابه مانند دستورات شرطی یا حلقه های تکرار.
3. **بین زیربرنامه ها:** ساختارهای استفاده شده بین زیربرنامه ها.

از دیدی دیگر می توان کنترل ترتیب دستورات را به 2 دسته تقسیم کرد:

1. **ضمنی (Implicit) :** منظور از ضمنی یعنی اینکه ترتیب اجرای دستورات به صورت پیش فرض توسط زبان برنامه سازی در نظر گرفته شده است.(مانند if)
2. **صریح (Explicit) :** یعنی اینکه کنترل ترتیب اجرای دستورات توسط برنامه نویس مشخص می شود مانند وارد کردن پرانتز در عبارات ریاضی یا استفاده از goto و... .

◄ کنترل ترتیب اجرا در داخل عبارات:

فرض کنید می خواهیم ریشه یک معادله درجه دوم را با فرمول زیر محاسبه نماییم :

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

برای محاسبه ی این فرمول کامپایلر حداقل 15 دستور را ایجاد می کند بنابراین بحثی که در اینجا مطرح است این است که ترتیب اجرای این 15 دستور چگونه است.

> ☑ در دسترس بودن عبارات در زبانهای سطح بالا یکی از فواید عمده ی آنها نسبت به زبانهای ماشین و اسمبلی است.

برای جوابگویی به سؤال فوق 2 روش وجود دارد :

روش اول: در این روش از یکی از 3 حالت infix,prefix,postfix استفاده می شود.

Syntax for Expression
Infix:
Prefix:
Postfix:

- **Infix** : این روش مناسب برای استفاده انسان ها می باشد همچنین برای زمانی کاربرد دارد که اعمال دودویی باشند(عملگرها، دو عملوند باشند.) بنابراین برای اعمالی با تعداد عملوندهای بیشتر باید از 2 روش دیگر استفاده شود.

نقاط ضعف infix:

1. اگر چه این روش مناسب به نظر می رسد ولی یک زبان برنامه سازی تنها نمی تواند شامل حالت infix باشد بلکه ترکیبی از infix و یکی از دو syntax دیگر را می بایست استفاده کرد.(مثلاً برای عملگرها بیشتر از دو عملوند infix کافی نیست)
2. می بایست اولویت عملگرها کاملاً مشخص شوند تا از ایجاد ابهام در عبارات ریاضی جلوگیری شود. مثلاً اگر a+b*c را داشته باشیم دو حالت مختلف پیش می آید :

مثلا : (a+b)*c که در زبان C و APL یکی است یا a+(b*c) (در زبان APL که دارای اولویت نیست.)

برای رفع این مشکل دو قانون زیر ارائه شده است :

1. انجام عملیات به صورت سلسله مراتبی و با رعایت اولویت به صورت زیر:

(+(یکانی),-(یکانی),not)تک عملوندها
^
*,/,%
+,-
<,<=,>,>=,<>,=
and,or,xor

2. در صورت داشتن دو عمل با اولویت مساوی شرکت پذیری می تواند از راست به چپ یا از چپ به راست باشد.

printf("……..",a+b,++a); : از راست به چپ در زبان C

برای مثال دستور printf و عبارات داخل آن در زبان C از اولویت راست به چپ استفاده می کند. توان هم در زبانی اگر پیاده سازی شود 2^{n^m} اولویت راست به چپ دارد.

- ☑ زبان APL بدون اولویت طراحی شده است و در آن عبارات از راست به چپ ارزیابی می شوند (یک شیوه از روش infix می باشد.)
- ☑ زبان smalltalk نیز مدلی مثل APL دارد با این تفاوت که عبارات از چپ به راست ارزیابی می شود.

♦ Prefix

به این روش لهستانی یا polish گفته می شود در این روش ابتدا عملگر و سپس عملوندها می آیند و به سه شکل می تواند موجود باشد:

1. ordinary : مثل *(+(A,B),-(C,A))
2. Cambridge polish : مثل (*(+AB)(-CA))
3. polish : مثل *+AB-CA

Snobol4 از روش prefix استفاده می کند.

✗ **مشکل روش polish (بدون پرانتز) این است** کـه دانـستن تعـداد عملونـدهای هـر عملگـر ضروری می باشد که این مشکل در دو روش اول آن رفع شده است.

✓ **مزیت polish معمولی در این است** که تبدیل آن به کد ماشین ساده تر است.

فراخوانی توابع معمولی در طراحی زبان از روش پیشوندی ساده استفاده می کنند.

♦ Postfix

در این روش عملگر بعد از عملوندها می آید نام دیگر آن RPN مـی باشـد. لهـستانی معکـوس دو روش دارد:

1. ordinary : ((A,B)+,(C,A)-)*
2. reverse polish : AB+CA-*

Postfix معمولی مشکل حالت prefix معمولی را ندارد چون در آن ابتدا عملوندها push می شوند.

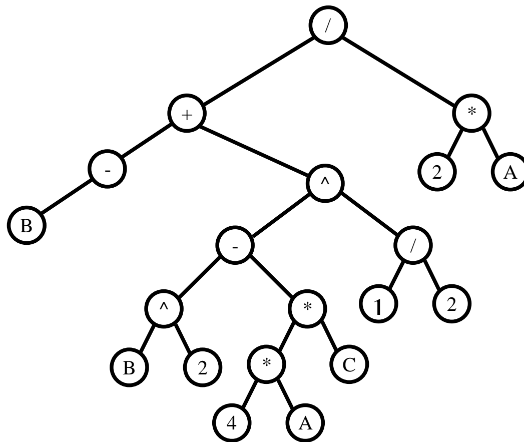
اکثر زبانها از روش postfix استفاده می کنند.

روش دوم: معرفی ساختار ریاضی به صورت ساختار درختی

در این روش ابتدا ساختار درختی عبارت ایجاد می شود و سپس درخت بر اساس یکی از پیمـایش های موجود پیمایش می شود و در انتها حاصل آن تعیین می گردد.

مثال :

R=(((-B)+((B^2-4*A*C)^(1/2)))/(2*A))



☑ زبان forth از روش postfix معمولی استفاده می کند و برای عمل کردن در کامپیوترهای کنترل فرآیندهای بلادرنگ طراحی شده است. این زبان امروزه به عنوان مبنایی برای postscript هنوز وجود دارد.

معرفی یک عبارت در زمان اجراء :

1. **prefix or postfix form** : در این روش الگوریتم پردازشگر عبارات را از سمت چپ به راست اسکن می کند و عملیات لازم را انجام می دهد. در بعضی از کامپیوترها این روش از طریق به کارگیری stack انجام می شود مانند snobol4 .
2. **tree structure** : در این روش درخت ایجاد شده بر اساس یکی از روشهای پیمایش پردازش می شود معمولاً زبانهای برنامه سازی مفسر نرم افزاری مانند زبان lisp از این روش استفاده می کند.
3. **machine code sequence** : در این روش عبارت به مجموعه ای از دستورات زبان ماشین تبدیل خواهد شد در ای حالت ممکن است محل های موقتی برای نتایج میانی مورد نیاز باشد. این روش به دلیل استفاده از سخت افزار کارایی بالایی دارد. مانند زبان ASM.

مشکلات ارزیابی عبارات :

1. ***قوانین ارزشیابی یکنواخت :*** به عنوان یک قاعده ی عمومی در درخت عبارات ابتدا عملوندها ارزشیابی می شوند و سپس عملگرها انجام می شوند این قاعده در اکثر اوقات کار نمی کند. برای مثال در زبان Algol و C داریم:

W:= z + (if x=0 then y else y/x) در زبانAlgol

W= z + ((x==0) ? y:y/x) در زبانC

در مثالهای فوق اگر قبل از انجام عملگرها ، عملوندها بررسی شوند به ازاء مقدار x=0 خطای تقسیم بر صفر پیش خواهد آمد.

قانون ارزشیابی فوق را که باعث ایجاد خطا در عبارات فوق می شود را قانون ارزشیابی سریع می نامند. برای برطرف کردن مشکل فوق یک قاعده ی دیگر به این صورت است که عملگر تصمیم می گیرد که آیا ارزشیابی عملوندها لازم است یا خیر یعنی در حقیقت عملگر ترتیب ارزشیابی عملوندها را مشخص می کند.

نکته: دو قانون ارزشیابی سریع و کند دو مکانیزم ارسال اطلاعات و پارامترها را به زیربرنامه پایه گذاری می کنند. نوع سریع پایه گذار call by value و نوع کند پایه گذار call by name و call by refrence است.

دو قانون ارزشیابی سریع و کند ، دو مکانیزم ارسال اطلاعات و پارامترها را به زیر برنامه پایه گذاری می کنند .

قانون ارزشیابی سریع ← Call By Value

قانون ارزشیابی کند ← Call By Reference / Call By Name

2. *اثرات جانبی* : فرض کنید تابعی وجود داشته باشد مانند fun(x) که در داخل آن مقدار a یک واحد افزایش یابد و سپس مقدار 3 برگردانده شود بنابراین تغییر متغیر a به هنگام اجرای ایـن تابع به عنوان اثر جانبی آن می باشد فرض کنید در ابتدا a ارزش یک داشته باشد حالا در عبارات زیر خروجی را مشخص کنید:

Y= A + fun(x) + A;

1) Y=2*3+2=8 بعد از ارزشیابی تابع مثل زبانC

2) Y=1*3+1=4 قبل از ارزشیابی تابع

3) Y=1*3+2=5 مثل زبانsmalltalk

4) Y=2*3+1=7 مثل زبانAPL

با توجه به مکانیزم های مختلف جهت ارزشیابی عبارات، ارزشیابی های متفاوتی می تواند انجام شود برای مثال :

حالت 1 در C : چون توابع اولویت بالایی دارند.

حالت 3 در smalltalk : از چپ به راست .

حالت 4 در APL : از راست به چپ .

استفاده از اعمالی که در عبارات دارای اثرات جانبی هستند پایـه گـذار تـضاد طـولانی در طراحـی زبانهای برنامه سازی بوده است. بعضی از زبانهای برنامه سازی بـه نـدرت اثـرات جـانبی را در یـک عبارت ایزوله می کنند و در اکثر زبانهای برنامه سـازی اثـرات جـانبی اجـازه داده مـی شـوند (بـا مشخص کردن ترتیب دقیق ارزشیابی عبارات).

مشکل اصلی اثرات جانبی این است که امکان بسیاری از انواع بهینه سازی ها وجود ندارد.

3. *شرایط خطا* : ایجاد شـرایط owerflow, underflow ، تقـسیم بـر صـفر و ... در حـین انجـام عملیات نمونه هایی از شرایط خطا هستند راه حل عمومی برای حل این مشکل وجود ندارد و از یک زبان به زبان دیگر متفاوت است.

4. *اتصال کوتاه در عبارات منطقی:*

مثال زیر را در نظر بگیرید:

If (A=0) OR (B/A>C) then

در مثال فوق اگر A=0 باشد و طبق قانون ارزشیابی سریع عمل نماییم در عبارت دوم خطای تقسیم بر صفر گرفته خواهد شد. در عبارات منطقی به روشی که همه ی عملوندها یا عبارات شرطی می بایست ارزیابی شوند اتصال بلند در عبارات منطقی گفته می شود که همانطور که مشاهده گردید در بعضی موارد باعث ایجاد خطا می شود. برای رفع مشکل فوق از قانون اتصال کوتاه در عبارات منطقی استفاده می شود به این ترتیب که در عبارات منطقی ، عملوندها یا عبارات شرطی یکی یکی ارزیابی شوند و عملیات تا جایی انجام شود که نیاز باشد. برای این حالت در مثال فوق اگر A=0 باشد پس از بررسی شرط اول دیگر نیازی به بررسی شرط دوم نمی باشد بنابراین خطا گرفته نمی شود.

از زبانهای C و Pascal معمولاً به صورت پیش فرض قانون مدار کوتاه انجام می شود ولی این قابلیت وجود دارد که اتصال بلند را نیز در آنها برقرار نماییم. در زبان Ada برای انجام AND و OR به صورت مدار کوتاه دو عملگر جدید Andelse و Orelse در نظر گرفته شده است. (بنابراین عملگرهای AND و OR برای اتصال بلند می باشند.)

مثال: while (I < UB) and (V(I) > 0) do

اگر شرط اول نادرست باشد (I > UB) شرط دوم خطا دارد (دسترسی به اندیس غیر مجاز در آرایه) در حالت اتصال بلند خطا دارد. در اتصال کوتاه مشکلی ندارد.

◄ کنترل ترتیب اجرا در بین دستورات:

انواع دستورات در یک دید ساده می توانند شامل موارد زیر باشند:

1. **Composition** (**ترکیبی**): کنترل ترتیب در ترکیبی از دستورات برنامه که به دنبال هم اجرا می شوند.
2. **Alternation** (**جیگزینی**) : دستوراتی که دو یا چند جایگزین یا جانشین جهت اجرا شدن در آنها وجود دارد. مثل : if … then … else…
3. **Iteration** (**تکراری**) : کنترل ترتیب در دستوراتی که برای چندین بار باید تکرار شوند مانند حلقه های تکرار.

♦ کنترل ترتیب با استفاده از GOTO

- پرش غیرشرطی

در این نوع دستور کنترل برنامه بدون شرط به محل دیگری از برنامه انتقال داده می شود .

```
goto L1
```

- پرش شرطی

این پرش در صورت برقراری یک شرط ، کنترل برنامه را به محل دیگری انتقال می دهد .

```
If (A=0) then goto L1;
```

با استفاده از goto می توان انواع کنترل در دستورات سه گانه Composition , Alernation , Itaration را انجام داد که به صورت زیر می باشد.

پیاده سازی انواع دستورات با goto

```
a) Composition
s0;
goto L1 ;
L2:s2 ;
Goto L3 ;
L1:s1 ;
Goto L2 ;
L3:s3 ;
```

اجرا :
s0,s1,s2,s3

```
b) Alternation
s0;
if a=0 then goto L1;
s1;
Goto L2;
L1:s2;
L2:s3;
```

اجرا :
T : s0,s2,s3
F : s0,s1,s3

```
c) Iteration
s0;
L1:if a=0 Goto L2;
S1;
Goto L1;
L2:s2;
```

اجرا :
T : s0,s2
F : s0,s1,s2

انواع Goto در زبانهای مختلف

سه نوع برچسب وجود دارد :

1. **برچسب ثابت** : label ها در این حالت به عنوان یک برچسب محلی و ثابت طی اجرای برنامه می باشند . این نوع از برچسب را تقریبا همه زبانهای برنامه سازی پشتیبانی می کنند . مثل C,Pascal
2. **برچسب بدون محاسبه** : در این نوع برچسب Label ها به عنوان داده محدود شده در طی اجرا می باشند و هیچ عملیات محاسباتی را نمی توان روی آنها انجام داد . در حقیقت در این روش Label ها به عنوان متغیرهایی هستند که می توانند به مکانهای مختلفی از برنامه مراجعه نمایند . مثل : Algol
3. **برچسب بدون محدودیت** : در این حالت Label ها به عنوان داده های محدود نشده در طی اجرا می باشند . در این نوع برچسب ها Label ها می توانند از ورودی خوانده شوند . حاصل یک عبارت به آنها انتساب شود و یا محاسبه ای نیز روی آنها انجام گیرد .
مثل : APL,Snobol4

برنامه سازی ساخت یافته و به کارگیری goto :

یکی از ویژگی های برنامه سازی ساخت یافته آن است که هر قسمت از برنامه تنها یک نقطه ورود و تنها یک نقطه خروج داشته باشد و یکی از راههای دستیابی بـه برنامـه نویـسی سـاختیافته ، عـدم استفاده از دستور goto می باشد .

+ مزایا

- ✓ استفاده مستقیم از سخت افزار و داشتن کارایی بالا در برنامه
- ✓ ساده و راحت برای برنامه های کوچک
- ✓ آشنا بودن برنامه نویس با آن مخصوصا برنامه نویسان اسمبلی
- ✓ توسط دستور goto یک بلاک از برنامه می تواند چندین هدف را سرویس دهد. .

> ☑ نشان داده شده است که حکم goto اضافی است در زبان ML اصلا دستور goto وجود ندارد .
> ☑ احکام break و continue شکلی از کنترل صریح و ساختیافته در C و Pascal می باشند (برخلاف goto)

– نقاط ضعف

- × ایجاد نقص و یا ضعیف کردن ساختار برنامه به صورت سلسه مراتبی یعنی استفاده از goto باعث می شود که ساختار ترتیبی برنامه به صورت یک ساختار سلسله مراتبـی درآیـد کـه فهم برنامه را مشکل می کند .
- × ترتیب دستورات در برنامه (در متن برنامه) ضرورتا تطابقی با ترتیب اجرا نخواهد داشت که این امر فهم برنامه را مشکل می کند .
- × مجموعه ای از دستورات ممکن است چندین هدف را سـرویس دهـد و ایـن خـلاف بحـث Single Purpose (SP) برنامه است که هر قسمت برنامه تنها یک عمل را انجام دهد .

♦ کنترل ترتیب ساختیافته

1. دستورات ترکیبی Composition

این نوع دستورات ، یک ترتیبی از دستورات می باشند که ممکن است مطابق یک دستور در برنامه های بزرگ عمل کنند و یا مجموعه ای از دستورات که داخل یک بلاک قرار می گیرند (از نظر پیاده سازی)

C,C++,Perl,Java

2. دستورات شرطی Alternation / Conditional Statement

- **دستورات IF**

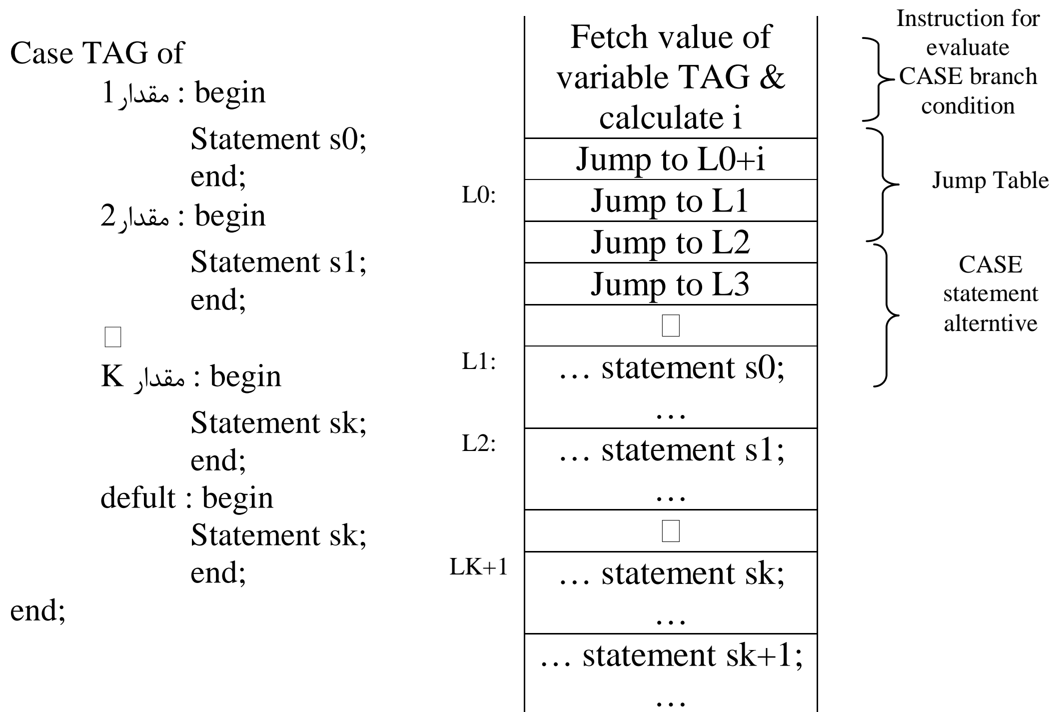
انواع :

1. single bramch — if (condition) {…}
2. two branch — if … / else …
3. multiple branch — if (condition) {…} else if (condition) {…}

- **دستور Case**

در واقع این دستور شکل دیگری از multiple branch است

برای پیاده سازی دستورات Case از یک jump table به صورت شکل زیر استفاده می شود .



☑ در دستور Case به دلیل ایجاد jump table در زمان کامپایل از elseif نردبانی سرعت کمتری دارد (در زمان کامپایل) ولی در زمان اجرا به دلیل وجود همین جدول از elseif نردبانی بهتر عمل می کند .

3. دستورات تکراری Interaction Statement

هر حلقه از دو قسمت تشکیل می شود :

Head : مشخص کننده تعداد تکرار Body است .

Body : شامل دستورات تکرار شونده است .

روش های موجود برای تکرار :

- **تکرار ساده** : در زبانهای cobol و Fortran

Perform body K times

در این مورد دو سئوال مطرح است :

1. آیا مقدار K فقط یکبار ارزشیابی می شود یا در حین اجرای body بـازهم مقـدار دهی می شود ؟
2. اگر مقدار K ، صفر یا منفی باشد آیا body یکبار اجرا می شود یا اصلا اجرا نمـی شود ؟

- **تکرار تا زمانی که یک شرط برقرار است** : در زبانهای C و Pascal

While (test) do body

- **تکرار در حالی که یک شمارنده افزایش می یابد** : در زبان Algol

For i:=1 step 2 until 30 do (body)

- **حلقه بی نهایت**

در جایی که شرایط خروج از حلقه پیچیده شد و به راحتی در head حلقه معمولی قابل بیان نیست از حلقه بی نهایت استفاده می شود .

Ada	**Pascal**	**C:**
loop	while (true)	for (; ;)
□	begin	{
exit when (condition)	□	□
	If condition then break;	If (condition) break;
end loop	□	□
	end;	}

جهت پیاده سازی از دستور jump/branch استفاده می شود .

- **تکرار بر مبنای داده ها** : گاهی اوقات داده ها مشخص کننده تعداد تکرار است . مـثلا در زبان perl :

for each $X (array item) { }

مشکلات در ساختار کنترل ترتیب :

1. **چندین راه خروج از حلقه**

باعث کاهش خوانایی برنامه شده و ساختیافتگی را از بین می برد .

2. **تکرار قسمتی از دستورات :**

در این حالت یک حلقه تکرار دارد که در همه حالات کل بدنه بـرای همـه دفعـات غیـر از دفعه آخر که از حلقه خارج می شویم اجرا می گردد .

3. **شرایط استثناء Exception Condition**

شرایط خطایی که در حـین اجـرا ممکـن اسـت بـه وجـود آیـد ماننـد تقـسیم بـر صـفر ، overflow ، ... که در این صورت کنترل برنامه را به هر قسمت دیگـری ممکـن اسـت منتقـل نماید .

◄ کنترل ترتیب بین زیربرنامه ها

برای بررسی این قسمت ابتدا از یک callreturn ساده با فرضیات زیر استفاده می کنیم .

1. زیربرنامه بازگشتی نباشد . ≠ زیربرنامه بازگشتی
2. یک فراخوانی به صورت صریح مورد نیاز است . ≠ شرایط استثناء
3. در هر بار call زیربرنامه ، دستورات آن به طور کامل اجرا شوند . ≠ Corutine
4. بلافاصله بعد از دستور call کنترل به زیربرنامه منتقل شود . ≠ زیربرنامه های زمانبدی شده
5. در هر لحظه از اجرا تنها یک زیربرنامه کنترل در دست بگیرد . ≠ کنترل همروند

☑ زبان فرترن این شرایط را در زیربرنامه ها دارد .

پیاده سازی

از جهت پیاده سازی باید در حالت زیر در نظر گرفته شود :

1. یک Activation شامل دو قسمت codesegment و activaition record بایـد وجـود داشته باشد .
2. قسمت codesegment در حین اجرا ثابت می باشد .

به ازای هر فراخوانی call یک AR به وجود می آید و در بالای پشته سیستم push می شـود و در ازای return آن AR ، pop شده و از بین می رود .

برای پیاده سازی به دو اشاره گر نیاز می باشد :

1. اشاره گر Current Instrument Pointer (CIP)

اشاره گری که به دستور در حال اجرا در کد اشاره می کند .

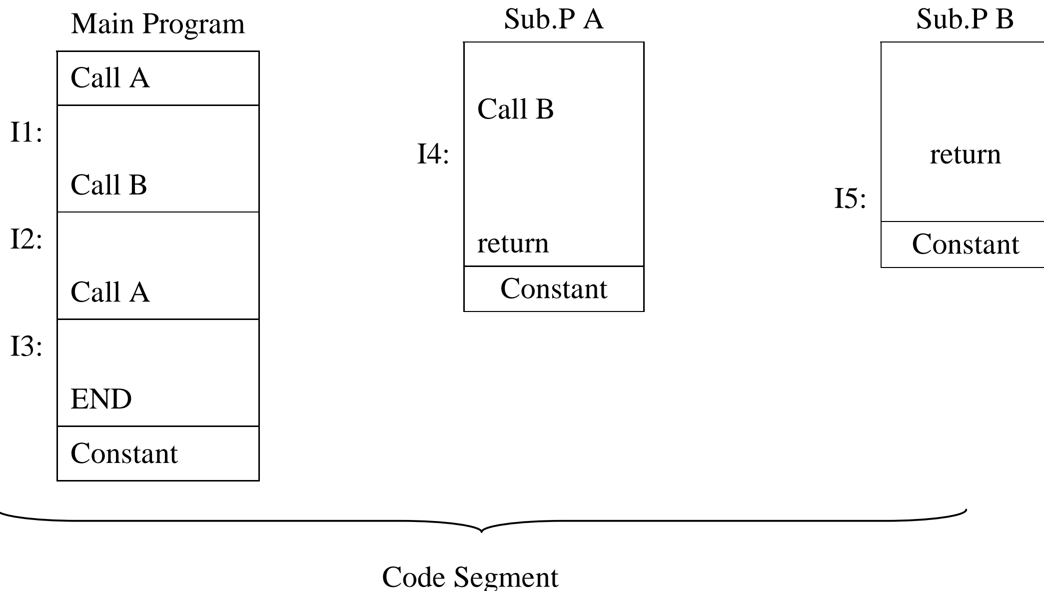
2. Current Environment Pointer (CEP)

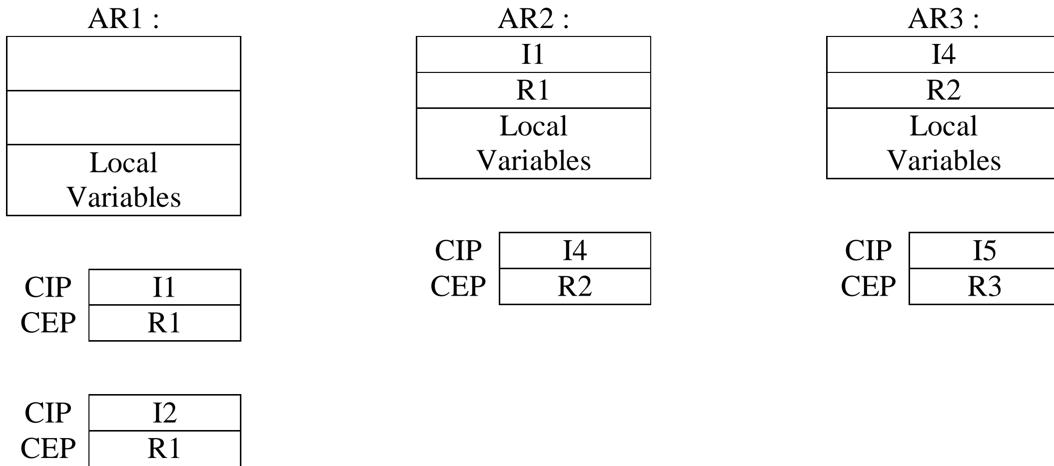
این اشاره گر به AR ای که فعال می شود اشاره می کند .

به ازای هر call یک AR ایجاد می شود و محتوای CIP و CEP برنامه فراخواننده در ابتـدای AR ذخیره می شود و در ازای return این دو اشاره گر return می شوند و کنترل برنامه ها دنبال می شود .

- ☑ به ازای فراخوانی هر زیربرنامه یک AR برای آن اختصاص می یابد که در آن عـلاوه بـر داده های محلی و پارامترها و ... برای آن زیربرنامه ، مقدار CIP و CEP برنامه فراخـوانی کننـده نیز در ابتدای آن ذخیره می شود . در هنگام return یک زیربرنامه نیز دوباره همین مقـادیر CIP و CEP ذخیره شده در AR بازیابی می شوند و برنامه ادامه کد را از سر می گیرد .
- ☑ پیشنهاد شده است که در قسمت .CodeSeg و AR در یک بلاک قرار گیرند که این امـر از جهت پیاده سازی کار را راحتتر می کند زیرا در این حالت CEP را می توان حـذف نمـود و AR اجاری انشعابی از قسمت CodeSeg مشخص شده توسط CIP می باشد که این امر در زبانهای Fortran و Cobol انجام شده است .
- ☑ CIP در زیربرنامه معادل PC در برنامه اصلی است .

مثال:





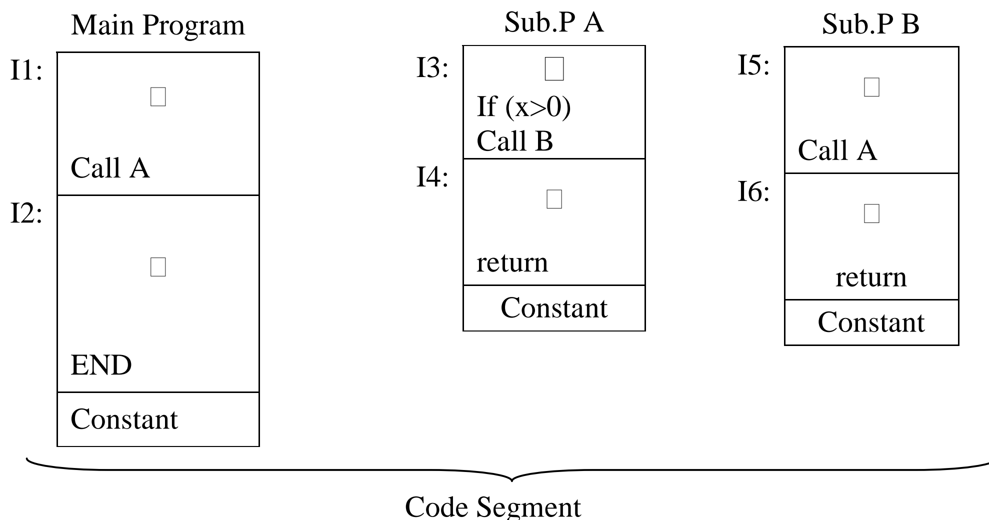
▼ زیر برنامه های بازگشتی

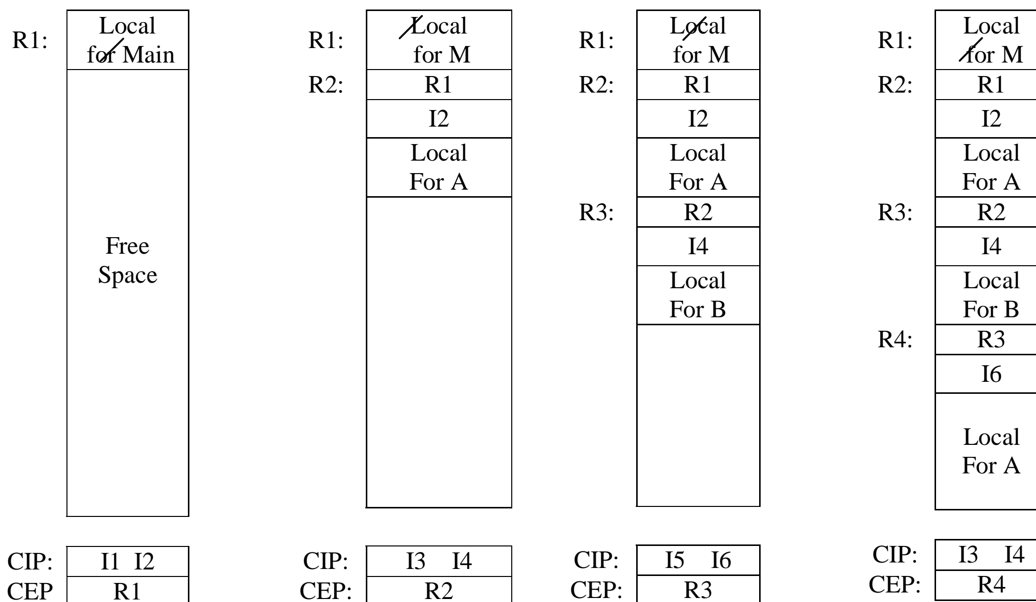
زیر برنامه هایی هستند که به طور مکرر خود را فراخوانی می کنند . تفاوت این زیر برنامه ها با زیر برنامه های بازگشتی در آن است که AR دومی و سومی و ... همه طی LifeTime اولین AR ایجاد می گردند .

از جهت پیاده سازی هر دو اشاره گر CIP و CEP باید استفاده شوند و از جهت چگونگی مدیریت آنها از پشته مرکزی استفاده می شود ؛ یعنی تمام AR ها به ترتیب فراخوانی ، در پشته ذخیره می شوند . در ازای هر call محتوای دو اشاره گر CIP و CEP در پشته ذخیره می شود و در ازای Return از پشته pop می شود.

طی عملیات کنترل ترتیب ، زنجیره ای از AR ها و یا لیستی از AR های روی پشته مرکزی پیاده سازی می شوند که این زنجیره از لینک ها را Dynamic Chain یا زنجیره پویا می گویند.

مثال :





- ☑ پشته معمولا برای زیر برنامه های بازگشتی استفاده می شود و زیربرنامه های ساده به آن نیازی ندارند . بنابراین باید مکانیزمی برای تشخیص بازگشتی از غیربازگشتی وجود داشته باشد . مثلا در PL1 زیربرنامه های بازگشتی با کلمه کلیدی Recursive مشخص می شوند و در زبانهای دیگری مثل C و Pascal همیشه ساختار بازگشتی در نظر گرفته می شوند یعنی همیشه از پشته استفاده می شود .
- ☑ چون زبان Pascal نسبت به مکان تعریف زیربرنامه حساس است ، بنابراین تنها از طریق دستور forward می تواند بازگشتی غیرمستقیم را پشتیبانی کند . Pckage در زبان Ada نیز از خواص مشابهی استفاده می کند .

مدیریت شرایط استثنایی که در ممکن است در برنامه ایجاد شود .

در حین اجرای زیربرنامه ها ممکن است عملیات خاصی برحسب ضرورت انجام شود . این شرایط خاص Exception در زیربرنامه های خاص پردازش این استثناء را Exception Handler می نامند .

انواع شرایط خاص

1. شرایط خطا مانند overflow که در حین انجام محاسبات به وجود می آید .
2. شرایطی که به طور غیرقابل پیش بینی در حین اجرا ممکن است بـه وجـود آینـد . ماننـد رسیدن به آخر فایل (EOF) به طور غیرمنتظره در هنگام کار با فایل
3. بررسی و Debug کردن

در حین Trace و Debug زیربرنامه ها بعضی شرایط خاص ممکن است به وجود آید ، زیرا ارزش متغیرها در زمان Debug ممکن است توسط برنامه نویس تغییر کند که در این گونه مواقع ، برنامه مترجم ، دستورات خاصی در بین دستورات تولید شده توسـط برنامـه بـرای Handle این شرایط و فراخوانی Handler قرار می دهد .

پیاده سازی برنامه های مربوط به Handle شرایط خاص

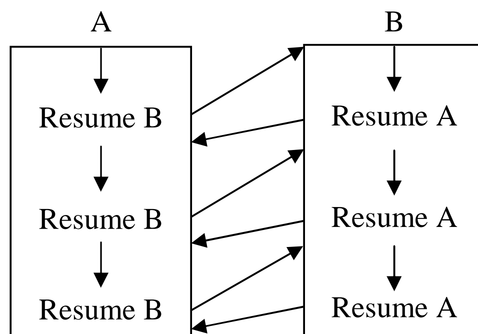
1. استفاده از Hardware Interrupt
2. استفاده از OS Interrupt یا نرم افزاری

سه حالت برای handle

- بعد از انجام عملیات خاص Handler ممکن است برنامه خاتمه یابد .
- بعد از انجام عملیات خـاص و دادن پیغـام خـاص ، کنتـرل بـه نقطـه ایجـاد Exception برگردانده شود.
- Exception ممکن است به زیربرنامه های دیگری انتشار یابد . (Propagate)

▼Coroutine

زیربرنامه هایی هستند که قبل از اجرای کامل می توانند به فراخوان برگردنـد . وقتـی همروالـی یـا Coroutine کنترل را از زیربرنامه ای دریافت کرد بخشی از آن اجرا می شود و هنگامی که کنترل را برمی گرداند ، اجرای آن به تعویق می افتد . برنامه فراخوان می تواند اجرای همـروال را از نقطـه ای که به تعویق افتاده است ، از سربگیرد (Resume)



کاربرد

- Simulation
- پردازش های RealTime
- Multiprocessing
- پردازش های موازی

پیاده سازی

از جهت پیاده سازی یک AR به صورت Static تخصیص می یابد و شامل CodeSegment و آدرس شروع خواهد بود .(بنابراین فقط CIP نیاز دارد)

در پیاده سازی Couroutine ها از پشته استفاده نمی شود . اگر زیربرنامه ای را Resume کردیم قبل از دنبال کردن کنترل آدرس فعلی را در ابتدای AR خودش ذخیره می کنیم و به محض Resume شدن couroutine نقطه شروع از ابتدای AR برداشته می شود و کنترل دنبال می گردد .

در هنگام Resume دو عمل انجام می شود :

1. آدرس فعلی Couroutine در حال اجرا در ابتدای AR خودش ذخیره می شود .
2. آدرس شروع Couroutine جدید از ابتدای AR آن برداشته می شود و زیربرنامه دنبال می شود .

	Coroutin A
	I1
I1:	Resume B Resume C Resume Main
	Constant
	Local for A

	Coroutin B
	I2
I2:	Resume C Resume A
	Constant
	Local for B

	Coroutin C
	I3
I3: I4:	Resume A
	Constant
	Local for C

▼زیربرنامه های برنامه ریزی شده (زمانبندی شده)

یکی دیگر از فرضیات مربوط به Call-Routine ساده این بود که بعد از فراخوانی زیربرنامه بلافاصله کنترل به آن زیربرنامه انتقال یابد . زیربرنامه های زمانبندی شده ، این قانون را نقض می کنند و کنترل طبق زمانبندی خاص در آنها انجام می گیرد .

انواع

1. یک زیربرنامه ممکن است قبل و یا بعد از زیربرنامه های دیگر ، کنترل له آن داده شود .

Call B After A;

2. کنترل زمانی به زیربرنامه داده می شود که شرایط خاصی برقرار شود .

Call B When (x=5) and (z>0);

3. فراخوانی زیربرنامه بر اساس مقیاس زمانی شبیه سازی شده.

Call B At Time – 30
Call B At Time – Current Time - 15

4. فراخوانی زیربرنامه برطبق یک اولویت دلخواه انجام می شود .

Call B With Prority 7

5. اگر زیربرنامه اولویتش از 7 بیشتر باشد ، قبل از فراخوانی B آن زیربرنامه اجرا می شود .

از جهت پیاده سازی

در برنامه Main بایستی لیست کاملی از زیربرنامه های زمان بندی شده همراه با مشخصات آنها (اولویت و ...) در آن تعریف شوند و کنترل تمام زیربرنامه ها از طریق Main انجام شود .