الگوی برنامه نویسی ۸۰۸۶/۸۸:

الگوی برنامه نویسی یک پردازنده شامل دو بخش است:

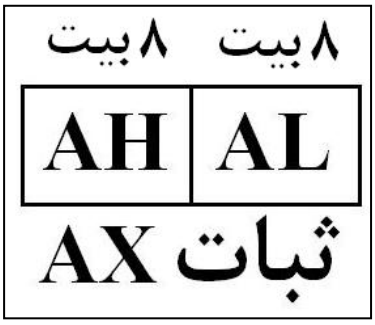
۱- آشنایی با معماری داخلی پردازنده: که شامل آشنایی با ثباتها، عملکرد و اندازه آنها و شیوه دسترسی پردازنده به حافظه و ورودی/خروجی هاست.

۲- آشنایی با قواعد زبان اسمبلی پردازنده

برای فراگیری مدل برنامه نویسی پردازنده ۸۰۸۶/۸۸ ابتدا به معماری داخلی آن می‌پردازیم:

ثباتهای ۸۰۸۶/۸۸:

الف) ثباتهای همه منظوره:



ثباتهای همه منظوره AX و BX ، CX و DX ثباتهای همه منظوره هستند که برای مقاصد مختلف قابل استفاده هستند. ویژگی منحصر به فرد این ثباتها قابلیت استفاده از آنها به صورت ۸ بیتی و ۱۶ بیتی است. مثلاً ثبات ۱۶ بیتی AX شامل یک بخش AH (۸ بیت بالایی) و AL (۸ بیت پایینی) است که می توان به هر بخش با نام آن مراجعه کرد.

در زیر با این ثباتها به صورت مختصر آشنا می‌شویم:

ثبات AX (Accumulator): ثبات انباره اصلی است و در دستورالعملهای ورودی/خروجی و محاسباتی مورد استفاده قرار می گیرد.

ثبات BX (Base): این ثبات به عنوان یک اندیس برای توسعه آدرس دهی و مراجعات جدولی به حافظه به کار می رود. کاربرد دیگر آن در انجام محاسبات است.

ثبات CX (Counter): از این ثبات معمولاً برای شمارش دفعات تکرار یک حلقه و نیز در محاسبات استفاده می شود.

ثبات DX (Data): از این ثبات در عملیات ورودی/خروجی به عنوان آدرس پورت و نیز در عملیاتهای ضرب و تقسیم با اعداد بزرگ استفاده می شود.

ب) ثبات پرچم:

در هر پردازنده یک ثبات به نام پرچم وجود دارد که بیتهای آن، وضعیت پردازنده بعد از انجام یک عمل را نشان می‌دهند. بیتهای ثبات پرچم ۱۶ بیتی ۸۰۸۶/۸۸ را در زیر می بینید:

--	--	--	--	O	D	I	T	S	Z	--	A	--	P	--	C

بیتهای خالی در ۸۰۸۶/۸۸ تعریف نشده اند. در زیر شرح مختصر بیتهای تعریف شده را می بینید.

بیت C (Carry): بیت رقم نقلی است. چنانچه یک عمل محاسباتی، تولید رقم نقلی کند (مثلاً جمع دو عدد بدون علامت که تولید سرریز کند) این بیت یک می شود. به‌علاوه این بیت حاوی رقم نقلی در عملیات منطقی شیفت نیز می باشد.

بیت D (Direction): بیت جهت است و توسط برنامه نویس برای کنترل جهت اعمال رشته ای مانند مقایسه یا انتقال رشته مقداردهی می شود.

بیت P (Parity): بیت توازن است و برای کنترل صحت اطلاعات ۸ بیت پایینی نتیجه یک عمل از طریق توازن فرد به کار می رود.

بیت A (Auxilary Carry): بیت رقم نقلی کمکی است و چنانچه در محاسبات ۸ بیتی رقم نقلی در بیت سوم ایجاد شود، برابر یک می شود. از این بیت در اعمال ریاضی BCD استفاده می شود.

بیت Z (Zero): بیت نشانگر نتیجه صفر است. چنانچه نتیجه یک عمل محاسباتی یا منطقی صفر باشد، این بیت برابر یک می شود.

بیت S (Sign): بیت علامت و منعکس‌کننده بیت علامت نتیجه آخرین عمل پردازنده است و از آن معمولاً برای تعیین علامت نتیجه استفاده می‌شود.

بیت T (Trap): بیت اجرای قدم به قدم است. چنانچه این بیت توسط برنامه نویس یک شده باشد، پردازنده بعد از اجرای هر دستور دچار وقفه خاصی می‌شود که نتیجه آن اجرای قطعه برنامه خاصی است که توسط برنامه نویس در مکان خاصی از حافظه نوشته شده و شامل دستوراتی برای نمایش محتویات ثباتهای داخلی و حافظه است.پس از استفاده کاربر از نتیجه‌این کار، با صدور فرمانی اجرای برنامه ادامه می‌یابد. از این بیت برای اشکال‌زدایی برنامه‌ها به کار می رود.

بیت I (Interrupt): بیت وقفه است و چنانچه توسط برنامه نویس صفر شده باشد، پردازنده به وقفه‌ها پاسخ نمی دهد.

بیت O (Overflow): بیت سرریز است و در صورت ایجاد سرریز در اعمال محاسباتی علامتدار، یک می شود.

ج) ثباتهای آدرس دهی:

مجموع ثباتهای سگمنت (CS، DS، SS و ES) و ثباتهای اشاره‌گر (BP، SP، IP) و ثباتهای اندیس (SI و DI) که همگی ۱۶ بیتی هستند، برای آدرس دهی حافظه به کار می روند.
پیش از مرور عملکرد آنها، نحوه مراجعه به حافظه در پردازنده ۸۰۸۶/۸۸ را بررسی می کنیم.

سگمنت[1] های حافظه:

برای مراجعه به حافظه در ۸۰۸۶/۸۸ (و پردازنده‌های بعدی اینتل)، حافظه به صورت قطعه‌هایی در نظر گرفته می‌شود که سگمنت نامیده می شوند. اندازه یک سگمنت حافظه حداکثر ۶۴ کیلوبایت است.

چهار نوع سگمنت حافظه وجود دارد که آنها را در ذیل بررسی می کنیم:

سگمنت کد (Code Segment): دستورالعملهای زبان ماشین که باید توسط پردازنده اجرا شوند در ناحیه ای از حافظه به نام سگمنت کد ذخیره می شوند؛ یعنی اولین دستور اجرایی در ابتدای این سگمنت قرار می گیرد. اگر کد برنامه بیش از ۶۴ کیلوبایت باشد، باید چند سگمنت کد در حافظه تعریف شود.

سگمنت داده (Data Segment): داده‌های مورد نیاز برنامه در این سگمنت قرار می گیرند. یک برنامه می تواند چند سگمنت داده داشته باشد.

سگمنت پشته (Stack Segment): اطلاعاتی که باید پیش از فراخوانی زیربرنامه‌ها در حافظه ذخیره شود تا بعد از بازگشت به برنامه اصلی بازیابی شود، در این سگمنت ذخیره می شود.

سگمنت اضافی (Extra Segment): این سگمنت در اعمال رشته ای مورد استفاده قرار می گیرد.

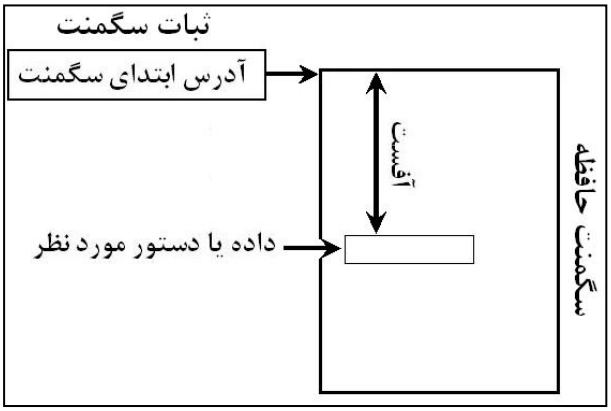
---

[1] *Segment*

ثباتهای سگمنت:

برای مراجعه به محتویات یک سگمنت باید آدرس آغاز آن سگمنت را بدانیم. ثباتهای ۱۶ بیتی سگمنت برای همین منظور به کار می روند:
ثبات CS آدرس ابتدای سگمنت کد، ثبات DS آدرس ابتدای سگمنت داده، ثبات SS آدرس ابتدای سگمنت پشته و ثبات ES آدرس ابتدای سگمنت اضافی را در خود نگه می دارند.

توجه به یک نکته ضروری است:
چون گذرگاه آدرس ۸۰۸۶/۸۸، ۲۰ بیتی است، قاعدتاً باید آدرس آغاز سگمنت‌ها که مکانهایی در حافظه هستند نیز ۲۰ بیتی باشد؛ پس چگونه آدرس آغاز یک سگمنت که ۲۰ بیتی است در ثبات ۱۶ بیتی سگمنت ذخیره می شود ؟!
آدرس آغاز یک سگمنت در حافظه که آدرسی ۲۰ بیتی است، باید همیشه آدرسی بخشپذیر بر ۱۶ باشد؛ مثل 00000 H یا F23E0 H یا 2A320 H یا ... . به همین دلیل همیشه ۴ بیت پایینی آدرس آغاز سگمنت صفر است و نیازی به ذخیره ندارد. مثلاً وقتی گفته می شود آدرس آغاز سگمنت کد که در CS ذخیره می شود، 2B3F H است، در واقع منظور آدرس ۲۰ بیتی 2B3F0 H است. بنابراین برای دستیابی به آدرس واقعی سگمنت، باید آن را ۴ بیت به سمت چپ شیفت دهیم.



ثباتهای اشاره گر و اندیس:

ثباتهای ۱۶ بیتی اشاره‌گر و اندیس حاوی فاصله دستورات و داده‌ها از مبدأ سگمنت هستند که اصطلاحاً آفست[1] نامیده می شود. منظور از آفست، فاصله آدرس داده یا دستورالعمل از ابتدای سگمنت آن است.

پرسش) با توجه به ۱۶ بیتی بودن ثباتهای آفست، نشان دهید که حداکثر اندازه یک سگمنت ۶۴ کیلوبایت است.

ثباتهای اشاره گر:

ثبات اشاره گر دستور (IP[2]): این ثبات حاوی فاصله دستور اجرایی بعدی از آغاز سگمنت کد است. آدرس دستورالعملها در برنامه‌نویسی به صورت CS:IP مشخص می شود؛ مثلاً وقتی به

[1] ***Offset***
[2] ***Instruction Pointer***

دستور واقع در مکان 2E35:0100 رجوع می شود، در واقع به دستوری که از مکان آغاز سگمنت کد (2E350) به اندازه 0100 واحد (۲۵۶ بایت) فاصله دارد، مراجعه می شود. آدرس این دستور در حافظه برابر 2E450 = 0100 + 2E350 است. به این آدرس، آدرس حقیقی یا فیزیکی و به آدرس IP آدرس منطقی گفته می شود.

ثبات اشاره گر مبنا (BP[1]): این ثبات حاوی فاصله‌ای در سگمنت پشته است و قابلیت دسترسی به محتویات پشته را برای برنامه نویس فراهم می آورد.

ثبات اشاره گر پشته (SP[2]): این ثبات حاوی فاصله ذخیره سازی داده‌ها در سگمنت پشته است و توسط پردازنده برای ذخیره و بازیابی اطلاعات در پشته مورد استفاده قرار می گیرد.

ثباتهای اندیس:

ثبات اندیس مبدأ (SI[3]): این ثبات در عملیات رشته ای برای نگهداری اندیس رشته منبع به کار می رود.

ثبات اندیس مقصد (DI[4]): این ثبات برای نگهداری اندیس رشته مقصد در عملیات رشته ای به کار می رود.

آدرسهای منطقی و فیزیکی:

آدرس داخل یک سگمنت می تواند از صفر تا FFFF (۶۵۵۳۵) تغییر کند. به آدرس داخل یک سگمنت، آدرس منطقی گفته می شود. مثلاً آدرس منطقی IP = 2F0A در سگمنت کد که CS = 204E معادل آدرس حقیقی (فیزیکی) ۲۰ بیتی 204E0 + 2F0A = 233EA در حافظه است. برای به دست آوردن آدرس حقیقی ۲۰ بیتی، آدرس شروع سگمنت را چهار بیت به سمت چپ شیفت داده و با آدرس آفست (آدرس منطقی) جمع می کنیم.

همانطور که می بینید آدرس حقیقی یک آدرس ۲۰ بیتی است که توسط پردازنده برای مراجعه به یک خانه حافظه روی گذرگاه آدرس گذاشته می شود؛ اما آدرس منطقی یک آدرس ۱۶ بیتی است که تنها درون محدوده ۶۴ کیلوبایتی یک سگمنت معنا دارد.

مهمترین مزیت استفاده از حافظه سگمنت‌بندی شده این است که برنامه‌هایی که تنها به آدرس منطقی ارجاع دهند را می توان در هر جای حافظه ذخیره و اجرا کرد؛ چون تغییر مکان ذخیره

---

[1] ***Base Pointer***
[2] ***Stack Pointer***
[3] ***Source Index***
[4] ***Destination Index***

برنامه فقط آدرسهای شروع سگمنت را تغییر می دهد که مستقل از آدرسهای منطقی است. بعنوان یک برنامه نویس به ندرت نیاز به دانستن آدرس فیزیکی یک مکان پیدا می کنید و معمولاً دانستن آدرس منطقی کافی است.

به این ترتیب مانند آنچه در سیستمهای عامل (مانند DOS و ویندوز) انجام می شود، می توان برای بهره برداری درست از فضای حافظه، انتخاب مکان بار کردن برنامه (مکان آغاز سگمنتها) را بدون نگرانی از درهم ریختن آدرسها به سیستم عامل سپرد. تنها شرط لازم، عدم مراجعه به آدرسهای فیزیکی و نیز عدم تغییر ثباتهای سگمنت در برنامه است.

ذخیره داده‌ها در حافظه:

۸۰۸۸ از نظر معماری داخلی مانند ۸۰۸۶ یک پردازنده ۱۶ بیتی است؛ یعنی گذرگاه داده داخلی آن ۱۶ بیتی است. اما گذرگاه داده خارجی آن (پینهای تراشه ۸۰۸۸ که به تبادل داده اختصاص یافته اند) ۸ بیتی است. سؤالی که به ذهن می آید این است که دستورات تبادل اطلاعات ۱۶ بیتی با خارج، چگونه در ۸۰۸۸ اجرا می شوند ؟

چون گذرگاه داده خارجی ۸۰۸۸ ۸ بیتی است، برای تبادل داده‌های ۱۶ بیتی چاره ای نیست جز اینکه داده‌های فوق را دو بار (در هر بار ۸ بیت) مبادله کنیم.

در پردازنده ۸۰۸۶ به دلیل ۱۶ بیتی بودن گذرگاه داده خارجی، این مشکل پیش نمی آید و ۸۰۸۶ می تواند هربار ۱۶ بیت اطلاعات را با خارج مبادله کند. به همین دلیل اجرای یک عملیات مشابه در ۸۰۸۸ کندتر از ۸۰۸۶ صورت می گیرد.

مشکلی که گذرگاه داده خارجی ۱۶ بیتی ۸۰۸۶ به وجود می آورد این است که تراشه‌های حافظه اکثراً دارای خانه‌های ۸ بیتی هستند، نه ۱۶ بیتی! شاید تحمیل حافظه‌های ۸ بیتی به یک پردازنده ۱۶ بیتی یک ضعف محسوب شود، اما حسن آن این است که کد ماشین بعضی دستورات، دارای تعداد فرد بایت (مثلاً یک یا ۳ یا ۵ بایت) است و پردازنده می‌تواند با آنها نیز به نحو مناسب رفتار کند.

در ۸۰۸۶ برای حل این مشکل، $۲^{۲۰}$ بایت فضای حافظه در دو بانک زوج و فرد در نظر گرفته می‌شود که ظرفیت هرکدام $۲^{۱۰}$ بایت است.

به یک نکته دیگر نیز دقت کنید:

هنگامی که اسمبلر، برنامه‌ای که حاوی عدد ۱۶ بیتی 2E34 است را به کد ماشین ترجمه می کند، در ترتیب کدهای ماشین ابتدا 34 و سپس 2E می آید و به‌همین ترتیب در حافظه ذخیره می‌شوند. پردازنده نیز هنگام واکشی یک عدد ۱۶ بیتی از حافظه به ترتیب عکس معمول عمل می‌کند؛ یعنی ابتدا ۸ بیت کم ارزش و سپس ۸ بیت پرارزش را دریافت می‌کند. چون پردازنده

اعداد ۱۶ بیتی را به صورت معکوس واکشی می کند، در نهایت عدد 2E34 به صورت واقعی در می‌آید.

واحدهای حافظه:

کوچکترین واحد اطلاعاتی در پردازنده‌ها، واحد بیت یعنی یک رقم دودویی (صفر یا یک) است. واحدهای دیگر Nibble (یک رشته ۴ بیتی) و بایت (یک رشته ۸ بیتی یا مجموع دو Nibble) هستند.

واحدهای اطلاعاتی دیگری نیز به صورت زیر تعریف می شوند:

کلمه (Word): یک قلم داده ۲ بایتی (۱۶ بیتی)

کلمه مضاعف (Double Word): یک قلم داده ۴ بایتی (۳۲ بیتی)

کلمه چهارگانه (Quad Word): یک قلم داده ۸ بایتی (۶۴ بیتی)

پاراگراف (Paragraph): یک قلم داده ۱۶ بایتی (۱۲۸ بیتی)

هر ۱۰۲۴ بایت یک کیلوبایت، هر ۱۰۲۴ کیلوبایت یک مگابایت و هر ۱۰۲۴ مگابایت یک گیگابایت نامیده می‌شود.

مثالهایی از برنامه‌نویسی به زبان اسمبلی ۸۰۸۶/۸۸:

مثال) قطعه برنامه ای بنویسید که **مرتباً** مقدار پورت ورودی شماره 27 H را خوانده و منفی آن را در پورت شماره 28 H نشان دهد.

```
AGAIN:  IN    AL,27H
        NEG   AL
        OUT   AL,28H
        JMP   AGAIN
```

چون عمل خواسته شده باید **مرتباً** انجام شود، آخرین دستور برنامه، دستور بازگشت به ابتدای برنامه باید باشد.

برای این کار به نقطه ابتدایی برنامه نام AGAIN را نسبت می دهیم. نام انتخاب شده دلخواه است و می تواند شامل حروف و ارقام باشد (اولین کاراکتر حتماً باید حرف باشد). بعد از نام انتخاب شده علامت ":" آورده می شود. این نام را برچسب[1] می‌نامند.

دستور JMP AGAIN، سبب انجام پرش به نقطه ای از برنامه به نام AGAIN که نقطه ابتدایی برنامه است، می‌شود. بنابراین اجرای قطعه برنامه فوق بی‌نهایت بار (تا زمانی که پردازنده خاموش یا بازنشانده نشده یا وقفه‌ای برای آن اتفاق نیفتاده است) ادامه می‌یابد.

---

[1] ***Label***

پرسش) الف) اگر در پایان برنامه ای دستور

HERE: JMP HERE

نوشته شود، با اجرای دستور فوق چه می شود ؟

ب) آیا می توان از این دستور به جای HALT استفاده کرد ؟

پرشهای شرطی و ساختارهای تصمیم:

دستور پرش غیر شرطی JMP در هر نقطه از برنامه که نوشته شود، سبب انجام پرش به نقطه ای دیگر از برنامه می شود.

برای تصمیم گیری در برنامه های اسمبلی از دستورات پرش شرطی استفاده می شود. این دستورات که تنوع زیادی نیز دارند، پس از یک دستور مقایسه مورد استفاده قرار می گیرند و بر اساس نتیجه مقایسه، تصمیم می گیرند. یک دستور پرش شرطی همیشه باعث پرش نمی شود، بلکه ابتدا یک شرط را بررسی کرده و در صورت صحیح بودن شرط، پرش انجام می شود؛ در غیر این صورت دستور بعد از دستور پرش شرطی اجرا خواهد شد.

قالب کلی استفاده از دستورات پرش شرطی به صورت زیر است:

CMP عملوند ۲ و عملوند ۱

دستور پرش شرطی به "مقصد پرش"

مجموعه دستورات ۱

JMP Next

مقصد پرش:

مجموعه دستورات ۲

Next: ----

دستور CMP[1] دستور مقایسه دو عملوند است و بر اساس این مقایسه در دستور پرش شرطی یک شرط بررسی می شود. در صورت صحت شرط پرش به "مقصد پرش" انجام شده و مجموعه دستورات ۲ اجرا می شود. در صورت صحیح نبودن شرط پرش انجام نشده و مجموعه دستورات ۱ اجرا شده و سپس با پرش به نقطه Next برنامه ادامه می یابد.

توجه کنید که دستور JMP Next ضروری است. اگر این دستور نباشد، بعد از مجموعه دستورات ۱، مجموعه دستورات ۲ نیز اجرا می شود! در حالی که می خواهیم در صورت صحیح نبودن شرط،

[1] ***Compare***

فقط مجموعه دستورات ۱ اجرا شود. بنابراین بعد از اجرای این دستورات به نقطه Next که ادامه برنامه است پرش انجام شده است. البته نقطه Next می تواند جایی قبل از دستور CMP نیز باشد.

پرسش) نشان دهید چنانچه بخواهیم در صورت صحیح نبودن شرط فقط مجموعه دستورات ۲ و در صورت صحیح بودن شرط مجموعه دستورات ۱ و ۲ اجرا شوند، نیازی به دستور JMP Next نیست.

همانطور که گفته شد، دستورات پرش شرطی تنوع زیادی دارند. برای آشنایی با کاربرد آنها چند مثال می آوریم.

مثال) قطعه برنامه ای بنویسید که مرتباً مقدار پورت 32H را بخواند و تا زمانی که مقدار آن برابر صفر نشده، عدد H 20 را در پورت 35H بنویسد و با صفر شدن آن، عدد FFH را در پورت نوشته و به اجرای برنامه پایان دهد.

```
AGAIN:      IN      AL,32H
            CMP     AL,0
            JZ      EXIT_PRGM
            MOV     AL,20H
            OUT     AL,35H
            JMP     AGAIN
EXIT_PRGM:  MOV     AL,0FFH
            OUT     AL,35H
            HALT
```

دستور CMP AL,0 ، محتویات ثبات AL (مقدار پورت ورودی 32H) را با صفر مقایسه می کند. این دستور تعدادی از بیتهای ثبات پرچم را تغییر می دهد که دستورات پرش شرطی براساس این بیتها تصمیم می گیرند.

دستور JZ EXIT_PRGM[1] در صورت صفر بودن نتیجه مقایسه، به جایی از برنامه به نام EXIT_PRGM پرش انجام می دهد. به جای دستور JZ، می توان از دستور JE[2] استفاده کرد. هردو دستور بر اساس بیت Z ثبات پرچم تصمیم می گیرند و در صورت یک بودن این بیت، پرش را انجام می دهند.

اگر AL صفر باشد، دستور JZ اجرا شده و پرش به EXIT_PRGM انجام می‌شود. اگر AL صفر نباشد، دستور بعد از JZ اجرا می شود که شامل ارسال 20H به پورت 35H و بازگشت به ابتدای برنامه برای خواندن مجدد پورت 32H است.

---

[1] ***Jump if Zero***
[2] ***Jump if Equal***

پرسش) این برنامه را با استفاده از دستور JNZ (یا JNE) بنویسید.

مثال) قطعه برنامه ای بنویسید که مرتباً پورتهای ورودی 40H و 41H را بخواند و اگر مقدار پورت 40H از پورت 41H بیشتر یا با آن مساوی بود، عدد 55H و در غیر این صورت عدد AA H را به پورت خروجی 30H ارسال کند.

```
AGAIN:      IN    AL,40H
            MOV   BL,AL
            IN    A,41H
            CMP   AL,BL
            JAE   SEND_AAH
            MOV   AL,55H
            OUT   AL,30H
            JMP   AGAIN
SEND_AAH:   MOV   AL,0AAH
            OUT   AL,30H
            JMP   AGAIN
```

دستور CMP AL,BL محتویات ثباتهای AL و BL را با هم مقایسه می کند.

دستور JAE SEND_AAH[1] در صورتی که عملوند اول دستور CMP (AL) بزرگتر یا مساوی عملوند دوم (BL) باشد، به مکانی به نام SEND_AAH پرش انجام می دهد که دستورات آن نقطه شامل ارسال AAH به پورت 30H و بازگشت به ابتدای برنامه است.

در صورتی که AL < BL، دستور JAE باعث پرش نمی شود و دستورات بعد از آن که شامل ارسال 55H به پورت 30H و بازگشت به ابتدای برنامه است، انجام می شود.

به جای دستور JAE، می توان از دستور JNB[2] نیز استفاده کرد.

ساختارهای تصمیم گیری:

در این بخش به طور مختصر ساختارهای تصمیم گیری مورد استفاده در زبانهای برنامه سازی سطح بالا را در زبان اسمبلی شبیه سازی می کنیم.

ساختار if-then-else:

در این ساختار ابتدا شرطی بررسی می شود و در صورت درست بودن شرط، مجموعه دستورات زیر if و در غیر این صورت مجموعه دستورات زیر else اجرا می شود:

---

[1] ***Jump if Above or Equal***

[2] ***Jump if Not Below***

if شرط then

مجموعه دستورات ۱

else

مجموعه دستورات ۲

در صورت صحیح بودن شرط، مجموعه دستورات ۱ و در غیر این صورت مجموعه دستورات ۲ اجرا می شود.

مثلاً دستورات زیر را در نظر بگیرید:

if sum > 100 then

مجموعه دستورات ۱

else

مجموعه دستورات ۲

معادل این دستورات در زبان اسمبلی به شرح زیر است:

```
CMP     sum,100
JA      P1
```

مجموعه دستورات ۲

```
JMP     end_if

P1:
```

مجموعه دستورات ۱

```
end_if: ------
```

پیاده‌سازی دیگر به صورت زیر است:

```
CMP     sum,100
JBE     P1
```

مجموعه دستورات ۱

```
JMP     end_if

P1:
```

مجموعه دستورات ۲

```
end_if: -------
```

ساختار switch (case):

در این ساختار مقدار یک متغیر با مقادیر ثابت مختلف مقایسه شده و در صورت تساوی با هرکدام عمل خاصی انجام می شود.

مثالی از ساختار switch در زبان C را در زیر می بینید:

```
switch (average) {
case  10:
```
مجموعه دستورات ۱
```
case  15:
```
مجموعه دستورات ۲
```
case  20:
```
مجموعه دستورات ۳
```
default:
```
مجموعه دستورات ۴
```
}
```

معادل اسمبلی ساختار بالا را در زیر می بینید:

```
CMP         average,10
JE          L1
CMP         average,15
JE          L2
CMP         average,20
JE          L3
JMP         L4
L1:
```
مجموعه دستورات ۱
```
JMP         Continue
L2:
```
مجموعه دستورات ۲
```
JMP         Continue
L3:
```
مجموعه دستورات ۳
```
JMP         Continue
L4:
```
مجموعه دستورات ۴
```
Continue: ----
```